Stochastic Processes and Simulations
A Machine Learning Perspective

Vincent Granville, Ph.D. | Version 6.0, June 2022

Sponsors

- MLTechniques. Private, self-funded Machine Learning research lab and publishing company. Developing explainable artificial intelligence, advanced data animations in Python including videos, model-free inference, and modern solutions to synthetic data generation. Visit our website, at MLTechniques.com.

Email the author at vincentg@MLTechniques.com to be listed as a sponsor.

Note: External links (in blue) and internal references (in red) are clickable throughout this document. Keywords highlighted in orange are indexed; those in red are both indexed and in the glossary section.

Contents

About this Textbook 2
Target Audience . . . 3
About the Author . . . 4

1 Poisson-binomial or Perturbed Lattice Process 5
1.1 Definitions . . . 6
1.2 Point Count and Interarrival Times . . . 7
1.3 Limiting Distributions, Speed of Convergence . . . 8
1.4 Properties of Stochastic Point Processes . . . 8
1.4.1 Stationarity . . . 8
1.4.2 Ergodicity . . . 9
1.4.3 Independent Increments . . . 9
1.4.4 Homogeneity . . . 9
1.5 Transforming and Combining Multiple Point Processes . . . 10
1.5.1 Marked Point Process . . . 10
1.5.2 Rotation, Stretching, Translation and Standardization . . . 10
1.5.3 Superimposition and Mixing . . . 11
1.5.4 Hexagonal Lattice, Nearest Neighbors . . . 12

2 Applications 12
2.1 Modeling Cluster Systems in Two Dimensions . . . 13
2.1.1 Generalized Logistic Distribution . . . 14
2.1.2 Illustrations . . . 15
2.2 Infinite Random Permutations with Local Perturbations . . . 16
2.3 Probabilistic Number Theory and Experimental Maths . . . 17
2.3.1 Poisson Limit of the Poisson-binomial Distribution, with Applications . . . 18
2.3.2 Perturbed Version of the Riemann Hypothesis . . . 20
2.4 Videos: Fractal Supervised Classification and Riemann Hypothesis . . . 22
2.4.1 Dirichlet Eta Function . . . 22
2.4.2 Fractal Supervised Classification . . . 23

3 Statistical Inference, Machine Learning, and Simulations 24
3.1 Model-free Tests and Confidence Regions . . . 24
3.1.1 Methodology and Example . . . 25
3.1.2 Periodicity and Amplitude of Point Counts . . . 28
3.1.3 A New Test of Independence . . . 30
3.2 Estimation of Core Parameters . . . 32
3.2.1 Intensity and Scaling Factor . . . 32
3.2.2 Model Selection to Identify F . . . 33
3.2.3 Theoretical Values Obtained by Simulations . . . 33
3.3 Hard-to-Detect Patterns and Model Identifiability . . . 34

3.4 Spatial Statistics, Nearest Neighbors, Clustering . . . 36
3.4.1 Stochastic Residues . . . 36
3.4.2 Inference for Two-dimensional Processes . . . 36
3.4.3 Clustering Using GPU-based Image Filtering . . . 40
3.4.4 Black-box Elbow Rule to Detect Outliers and Number of Clusters . . . 41
3.5 Boundary Effect . . . 44
3.5.1 Quantifying some Biases . . . 44
3.5.2 Extreme Values . . . 46
3.6 Poor Random Numbers and Other Glitches . . . 48
3.6.1 A New Type of Pseudo-random Number Generator . . . 48

4 Theorems **49**
4.1 Notations . . . 49
4.2 Link between Interarrival Times and Point Count . . . 49
4.3 Point Count Arithmetic . . . 50
4.4 Link between Intensity and Scaling Factor . . . 50
4.5 Expectation and Limit Distribution of Interarrival Times . . . 51
4.6 Convergence to the Poisson Process . . . 51
4.7 The Inverse or Hidden Model . . . 52
4.8 Special Cases with Exact Formula . . . 53
4.9 Fundamental Theorem of Statistics . . . 54

5 Exercises, with Solutions **55**
5.1 Full List . . . 55
5.2 Probability Distributions, Limits and Convergence . . . 55
5.3 Features of Poisson-binomial Processes . . . 59
5.4 Lattice Networks, Covering Problems, and Nearest Neighbors . . . 61
5.5 Miscellaneous . . . 65

6 Source Code, Data, Videos, and Excel Spreadsheets **69**
6.1 Interactive Spreadsheets and Videos . . . 70
6.2 Source Code: Point Count, Interarrival Times . . . 71
6.2.1 Compute $\text{E}[N(B)], \text{Var}[N(B)]$ and $P[N(B) = 0]$ . . . 72
6.2.2 Compute $\text{E}[T], \text{Var}[T]$ and $\text{E}[T^r]$ . . . 73
6.2.3 Produce random deviates for various F's . . . 73
6.2.4 Compute $F(x)$ for Various F . . . 73
6.3 Source Code: Radial Cluster Simulation . . . 74
6.4 Source Code: Nearest Neighbor Distances . . . 75
6.5 Source Code: Detection of Connected Components . . . 78
6.6 Source Code: Visualizations, Density Maps . . . 81
6.6.1 Visualizing the Nearest Neighbor Graph . . . 81
6.6.2 Clustering and Density Estimation via Image Filtering . . . 82
6.7 Source Code: Production of the Videos . . . 85
6.7.1 Dirichlet Eta Function . . . 85
6.7.2 Fractal Supervised Clustering . . . 86

Glossary **89**

List of Figures **90**

References **91**

Index **94**

About this Textbook

This scratch course on stochastic processes covers significantly more material than usually found in traditional books or classes. The approach is original: I introduce a new yet intuitive type of random structure called perturbed lattice or Poisson-binomial process, as the gateway to all the stochastic processes. Such models have started to gain considerable momentum recently, especially in sensor data, cellular networks, chemistry, physics and engineering applications. I present state-of-the-art material in simple words, in a compact style, including new research developments and open problems. I focus on the methodology and principles, providing the reader with solid foundations and numerous resources: theory, applications, illustrations, statistical inference, references, glossary, educational spreadsheet, source code, stochastic simulations, original exercises, videos and more.

Below is a short selection highlighting some of the topics featured in the textbook. Some are research results published here for the first time.

GPU clustering	Fractal supervised clustering in GPU (graphics processing unit) using image filtering techniques akin to neural networks, automated black-box detection of the number of clusters, unsupervised clustering in GPU using density (gray levels) equalizer
Inference	New test of independence, spatial processes, model fitting, dual confidence regions, minimum contrast estimation, oscillating estimators, mixture and superimposed models, radial cluster processes, exponential-binomial distribution with infinitely many parameters, generalized logistic distribution
Nearest neighbors	Statistical distribution of distances and Rayleigh test, Weibull distribution, properties of nearest neighbor graphs, size distribution of connected components, geometric features, hexagonal lattices, coverage problems, simulations, model-free inference
Cool stuff	Random functions, random graphs, random permutations, chaotic convergence, perturbed Riemann Hypothesis (experimental number theory), attractor distributions in extreme value theory, central limit theorem for stochastic processes, numerical stability, optimum color palettes, cluster processes on the sphere
Resources	28 exercises with solution expanding the theory and methods presented in the textbook, well documented source code and formulas to generate various deviates and simulations, simple recipes (with source code) to design your own data animations as MP4 videos – see ours on YouTube

This first volume deals with point processes in one and two dimensions, including spatial processes and clustering. The next volume in this series will cover other types of stochastic processes, such as Brownian-related and random, chaotic dynamical systems. The point process which is at the core of this textbook is called the Poisson-binomial process (not to be confused with a binomial nor a Poisson process) for reasons that will soon become apparent to the reader. Two extreme cases are the standard Poisson process, and fixed (non-random) points on a lattice. Everything in between is the most exciting part.

Target Audience

College-educated professionals with an analytical background (physics, economics, finance, machine learning, statistics, computer science, quant, mathematics, operations research, engineering, business intelligence), students enrolled in a quantitative curriculum, decision makers or managers working with data scientists, graduate students, researchers and college professors, will benefit the most from this textbook. The textbook is also intended to professionals interested in automated machine learning and artificial intelligence.

It includes many original exercises requiring out-of-the-box thinking, and offered with solution. Both students and college professors will find them very valuable. Most of these exercises are an extension of the core material. Also, a large number of internal and external references are immediately accessible with one click, throughout the textbook: they are highlighted respectively in red and blue in the text. The material is organized to facilitate the reading in random order as much as possible and to make navigation easy. It is written for busy readers.

The textbook includes full source code, in particular for simulations, image processing, and video generation. You don't need to be a programmer to understand the code. It is well documented and easy to read, even for people with little or no programming experience. Emphasis is on good coding practices. The goal is to help you quickly develop and implement your own machine learning applications from scratch, or use the ones offered in the textbook. The material also features professional-looking spreadsheets allowing you to perform interactive statistical tests and simulations in Excel alone, without statistical tables or any coding. The code, data sets, videos and spreadsheets are available on my GitHub repository.

The content in this textbook is frequently of graduate or post-graduate level and thus of interest to researchers. Yet the unusual style of the presentation makes it accessible to a large audience, including students and professionals with a modest analytic background (a standard course in statistics). It is my hope that it will entice beginners and practitioners faced with data challenges, to explore and discover the beautiful and useful aspects of the theory, traditionally inaccessible to them due to jargon.

About the Author

Vincent Granville, PhD is a pioneering data scientist and machine learning expert, co-founder of Data Science Central (acquired by a publicly traded company in 2020), former VC-funded executive, author and patent owner. Vincent's past corporate experience includes Visa, Wells Fargo, eBay, NBC, Microsoft, CNET, InfoSpace and other Internet startup companies (one acquired by Google). Vincent is also a former post-doct from Cambridge University, and the National Institute of Statistical Sciences (NISS). He is currently publisher at DataShaping.com. He makes a living as an independent researcher working on stochastic processes, dynamical systems, experimental math and probabilistic number theory.

Vincent published in Journal of Number Theory, Journal of the Royal Statistical Society (Series B), and IEEE Transactions on Pattern Analysis and Machine Intelligence, among others. He is also the author of multiple books, including "Statistics: New Foundations, Toolbox, and Machine Learning Recipes", "Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems" with a combined reach of over 250,000, as well as "Becoming a Data Scientist" published by Wiley. For details, see my Google Scholar profile, here.

1 Poisson-binomial or Perturbed Lattice Process

I introduce here one of the simplest point process models. The purpose is to illustrate, in simple English, the theory of point processes using one of the most elementary and intuitive examples, keeping applications in mind. Many other point processes will be covered in the next sections, both in one and two dimensions. Key concepts, soon to be defined, include:

Category	Description	Book sections
Top parameters	Intensity λ – granularity of the process	4.4, 3.2.1
	Scaling factor s – quantifies point repulsion or mixing	3.1.1, 3.2.1
	Distribution F – location-scale family, with $F_s(x) = F(x/s)$	1.1, 3.2.2
Properties	Stationarity and ergodicity	1.4, 5.3
	Homogeneity and anisotropy	1.4.4
	Independent increments	1.4.3, 3.1.3
Core distributions	Interarrival times T	1.2, 4.2
	Nearest neighbor distances	3.4, 5.4
	Point count $N(B)$ in a set B	4.3, 5.3
	Point distribution (scattering, on a set B)	1.2
Type of process	Marked point process	1.5.1
	Cluster point process	2.1, 2.1.2
	Mixtures and interlacings (superimposed processes)	1.5.3, 3.4.3
Topology	Lattice space (index space divided by λ)	2.1, 4.7
	State space (where the points are located)	2.1
	Index space (hidden space of point indices: $\mathbb{Z}$ or $\mathbb{Z}^2$)	4.7, 2.2
Other concepts	Convergence to stationary Poisson point process	1.3, 4.6
	Boundary effects	3.5
	Dimension (of the state space)	1.2
	Model identifiability	3.3

I also present several probability distributions that are easy to sample from, including logistic, uniform, Laplace and Cauchy. I use them in the simulations. I also introduce new ones such as the exponential-binomial distribution (the distribution of interarrival times), and a new type of generalized logistic distribution. One of the core distributions is the Poisson-binomial with an infinite number of parameters. The Poisson-binomial process is named after that distribution, attached to the point count (a random variable) counting the number of points found in any given set. By analogy, the Poisson point process is named after the Poisson distribution for its point count. Poisson-binomial processes are also known as perturbed lattice point processes. Lattices, also called grids, are a core topic in this textbook, as well as nearest neighbors.

Poisson-binomial processes are different from both Poisson and binomial processes. However, as we shall see and prove, they converge to a Poisson process when a parameter called the scaling factor (closely related to the variance), tends to infinity. In recent years, there has been a considerable interest in perturbed lattice point processes, see [62, 68]. The Poisson-binomial process is lattice-based, and indeed, perturbed lattice point processes and Poisson-binomial processes are one and the same. The name "Poisson-binomial" has historical connotations and puts emphasis on its combinatorial nature, while "perturbed lattice" is more modern, putting emphasis on topological features and modern applications such as cellular networks.

Poisson-binomial point processes with small scaling factor s are good at modeling lattice-based structures such as crystals, exhibiting repulsion (also called inhibition) among the points, see Figure 3. They are also widely used in cellular networks, see references in Section 2.1.

1.1 Definitions

A point process is a (usually infinite) collection of points, sometimes called events in one dimension, randomly scattered over the real line (in one dimension), or over the entire space in higher dimensions. The points are denoted as X_k with $k \in \mathbb{Z}$ in one dimension, or (X_h, X_k) with $(h, k) \in \mathbb{Z}^2$ in two dimensions. The random variable X_k takes values in $\mathbb{R}$, known as the state space . In two dimensions, the state space is $\mathbb{R}^2$. The points are assumed to be independently distributed, though not identically distributed. Later in this textbook, it will be evident from the context when we are dealing with the one or two dimensional case.

In one dimension, the Poisson-binomial process is characterized by infinitely many points X_k, $k \in \mathbb{Z}$, each centered around k/λ, independently distributed with

$$P(X_k < x) = F\Big(\frac{x - k/\lambda}{s}\Big), \tag{1}$$

where

- The parameter $\lambda > 0$ is called the intensity; it represents the granularity of the process. The expected number of points in an interval of length $1/\lambda$ (in one dimension) or in a square of area $1/\lambda^2$ (in two dimensions), is equal to one. This generalizes to higher dimensions. The set $\mathbb{Z}/\lambda$ (or $\mathbb{Z}/\lambda \times \mathbb{Z}/\lambda$ in two dimensions) is the underlying lattice space of the process (also called the grid), while $\mathbb{Z}$ (or $\mathbb{Z}^2$ in two dimensions) is called the index space. The difference between state and lattice space is illustrated in Figure 22.
- The parameter $s > 0$ is the scaling factor, closely related to the variance. It determines the degree of mixing among the X_k's. When $s = 0$, $X_k = k/\lambda$ and the points are just the lattice points; there is no randomness. When s is infinite, the process becomes a classic stationary Poisson point process of intensity λ^d, where d is the dimension.
- The cumulative distribution function (CDF) $F(x)$ is continuous and belongs to a family of location-scale distributions [Wiki]. It is centered at the origin ($F(0) = \frac{1}{2}$), and symmetric ($F(x) = 1 - F(-x)$). Thus it has zero expectation, assuming the expectation exists. Its derivative, denoted as $f(x)$, is the density function; it is assumed to be unimodal (it has only one maximum), with the maximum value attained at $x = 0$.

In two dimensions, Formula (1) becomes

$$P[(X_h, Y_k) < (x, y)] = F\Big(\frac{x - h/\lambda}{s}\Big) F\Big(\frac{y - k/\lambda}{s}\Big). \tag{2}$$

Typical choices for F are

$$\begin{aligned}
&\text{Uniform: } F(x) = \frac{1}{2} + \frac{x}{2} \text{ if } -1 \leq x \leq 1, \text{ with } F(x) = 1 \text{ if } x > 1 \text{ and } F(x) = 0 \text{ if } x < -1\\
&\text{Laplace: } F(x) = \frac{1}{2} + \frac{1}{2}\text{sgn}(x)(1 + \exp(-|x|))\\
&\text{Logistic: } F(x) = \frac{1}{1 + \exp(-x)}\\
&\text{Cauchy: } F(x) = \frac{1}{2} + \frac{1}{\pi}\arctan(x)
\end{aligned}$$

where $\text{sgn}(x)$ is the sign function [Wiki], with $\text{sgn}(0) = 0$. Despite the appearance, I use the standard form of these well-known distributions, when the location parameter is zero, and the scaling factor is $s = 1$. It looks unusual because I define them via their cumulative distribution function (CDF), rather than via the more familiar density function. Throughout this textbook, I use the CDF and its inverse (the quantile function) for simulation purposes.

F	Uniform	Logistic	Laplace	Cauchy	Gaussian
$\text{Var}[F_s]$	$s^2/3$	$\pi^2 s^2/3$	$2s^2$	∞	s^2

Table 1: Variance attached to F_s, as a function of s

Table 1 shows the relationship between s and the actual variance, for the distributions in question. I use the notation $F_s(x) = F(x/s)$ and $f_s(x)$ for its density, interchangeably throughout this textbook. Thus, $F(x) = F_1(x)$ and $f(x) = f_1(x)$. In orther words, F is the standardized version of F_s. In two dimensions, I use

$F(x,y) = F(x)F(y)$, assuming independence between the two coordinates: see Formula (2).

Remark: The parameter s is called the scaling factor because it is proportional to the variance of F_s, but visually speaking, it represents the amount of repulsion among the points of the process. See visual impact of a small s in Figure 3, and of a larger one in Figure 4.

1.2 Point Count and Interarrival Times

An immediate result is that $F_s(x - k/\lambda)$ is centered at k/λ. Also, if $s = 0$, then $X_k = k/\lambda$. If s is very small, X_k is very close to k/λ most of the time. But when s is large, the points X_k's are no longer ordered, and the larger s, the more randomly they are permutated (or shuffled, or mixed) on the real line.

Let $B = [a, b]$ be an interval on the real line, with $a < b$, and $p_k = P(X_k \in B)$. We have:

$$\begin{aligned} p_k &= F_s(b - t_k) - F_s(a - t_k) \\ &= F\Big(\frac{b - k/\lambda}{s}\Big) - F\Big(\frac{a - k\lambda}{s}\Big) \end{aligned} \tag{3}$$

This easily generalizes to two dimensions based on Formula (2). As a consequence, the integer-valued random variable $N(B)$ counting the number of points of the process in a set B, known as the counting measure [Wiki] or point count , has a Poisson-binomial distribution of parameters $p_k, k \in \mathbb{Z}$ [Wiki]. The only difference with a standard Poisson-binomial distribution is that here, we have infinitely many parameters (the p_k's). Basic properties of that distribution yield:

$$\text{E}[N(B)] = \sum_{k=-\infty}^{\infty} p_k \tag{4}$$

$$\text{Var}[N(B)] = \sum_{k=-\infty}^{\infty} p_k(1 - p_k) \tag{5}$$

$$P[N(B) = 0] = \prod_{k=-\infty}^{\infty} (1 - p_k) \tag{6}$$

$$P[N(B) = 1] = \Big(\sum_{k=-\infty}^{\infty} \frac{p_k}{1 - p_k}\Big) \cdot P[N(B) = 0] \tag{7}$$

It is more difficult, though possible, to obtain the higher moments $\text{E}[N^r(B)]$ or $P[N(B) = r]$ in closed form if $r > 2$. This is due to the combinatorial nature of the Poisson-binomial distribution. But you can easily obtain approximated values using simulations.

Another fundamental, real-valued random variable, denoted as T or $T(\lambda, s)$, is the interarrival times between two successive points of the process, once the points are ordered on the real line. In two dimensions, it is replaced by the distance between a point of the process, and its nearest neighbor. Thus it satisfies (see Section 4.2) the following identity:

$$P(T > y) = P[N(B) = 0],$$

with $B =]X_0, X_0 + y]$, assuming it is measured at X_0 (the point of the process corresponding to $k = 0$). See Formula (38) for the distribution of T. In practice, this intractable exact formula is not used; instead it is approximated via simulations. Also, the point X_0 is not known, since the X_k's are in random order, and retrieving k knowing X_k is usually not possible. The indices (the k's) are hidden. However, see Section 4.7. The fundamental question is whether using X_0 or any X_k (say X_5), matters for the definition of T. This is discussed in Section 1.4 and illustrated in Table 4.

Finally, the point distribution is also of particular interest. In one dimension, this distribution can be derived from the distribution of interarrival times: the distance between two successive points. For instance, for a stationary Poisson process on the real line (that is, the intensity λ does not depend on the location), the points in any given set B are uniformly and independently distributed in B, and the interarrival times have an exponential distribution of expectation $1/\lambda$. However, for Poisson-binomial processes, there is no such simple result. If s is small, the points are more evenly spaced than the laws of pure randomness would dictate, see Figure 3. Indeed, the process is called repulsive: it looks as if the points behave like electrical charges, all of the same sign, exercising repulsive forces against each other. Despite this fact, the points are still independently distributed. To the contrary, cluster processes later investigated in this textbook, exhibit point attraction: it looks as if the points are attracted to each other.

Remark: A binomial process is defined as a finite set of points uniformly distributed over a domain B of finite area. Usually, the number of points is itself random, typically with a binomial distribution.

1.3 Limiting Distributions, Speed of Convergence

I prove in Theorem 4.5 that Poisson-binomial processes converge to ordinary Poisson processes. In this section, I illustrate the rate of convergence, both for the interarrival times and the point count in one dimension.

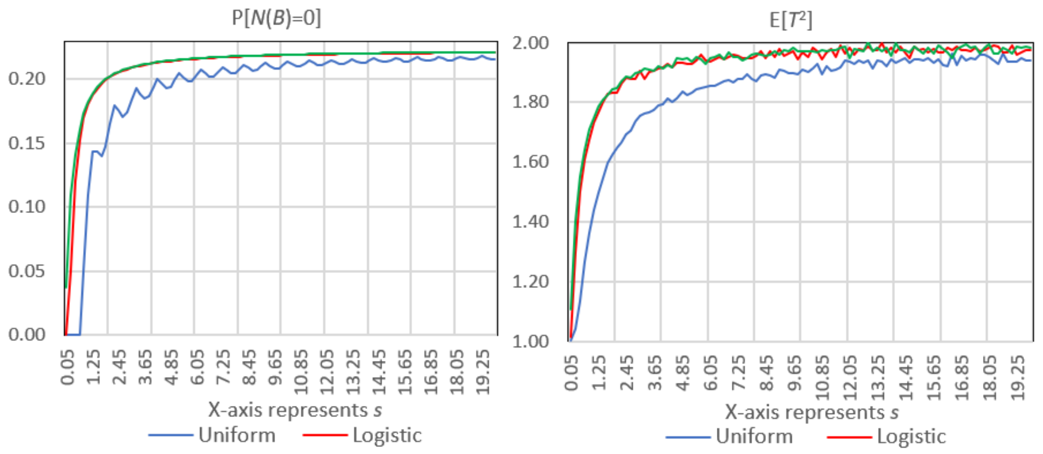

Figure 1: Convergence to stationary Poisson point process of intensity λ

In Figure 1, we used $\lambda = 1$ and $B = [-0.75, 0.75]$; $\mu(B) = 1.5$ is the length of B. The limiting values (combined with those of Table 3), as $s \to \infty$, are in agreement with $N(B)$'s moments converging to those of a Poisson distribution of expectation $\lambda\mu(B)$, and T's moments to those of an exponential distribution of expectation $1/\lambda$. In particular, it shows that $P[N(B) = 0] \to \exp[-\lambda\mu(B)]$ and $E[T^2] \to 2/\lambda$ as $s \to \infty$. These limiting distributions are features unique to stationary Poisson processes of intensity λ.

Figure 1 illustrates the speed of convergence of the Poisson-binomial process to the stationarity Poisson process of intensity λ, as $s \to \infty$. Further confirmation is provided by Table 3, and formally established by Theorem 4.5. Of course, when testing data, more than a few statistics are needed to determine whether you are dealing with a Poisson process or not. For a full test, compare the empirical moment generating function (the estimated $\mathrm{E}[T^r]$'s say for all $r \in [0, 3]$) or the empirical distribution of the interarrival times, with its theoretical limit (possibly obtained via simulations) corresponding to a Poisson process of intensity λ. The parameter λ can be estimated based on the data. See details in Section 3.

In Figure 1, the values of $\mathrm{E}[T^2]$ are more volatile than those of $P[N(B) = 0]$ because they were estimated via simulations; to the contrary, $P[N(B) = 0]$ was computed using the exact Formula (6), though truncated to 20,000 terms. The choice of a Cauchy or logistic distribution for F makes almost no difference. But a uniform F provides noticeably slower, more bumpy convergence. The Poisson approximation is already quite good with $s = 10$, and only improves as s increases. Note that in our example, $N(B) > 0$ if $s = 0$. This is because $X_k = k$ if $s = 0$; in particular, $X_0 = 0 \in B = [-0.75, 0.75]$. Indeed $N(B) > 0$ for all small enough s, and this effect is more pronounced (visible to the naked eye on the left plot, blue curve in Figure 1) if F is uniform. Likewise, $E[T^2] = 1$ if $s = 0$, as $T(\lambda, s) = \lambda$ if $s = 0$, and here $\lambda = 1$.

The results discussed here in one dimension easily generalize to higher dimensions. In that case B is a domain such as a circle or square, and T is the distance between a point of the process, and its nearest neighbor. The limit Poisson process is stationary with intensity λ^d, where d is the dimension.

1.4 Properties of Stochastic Point Processes

In this section, we review key features of point processes in general, applied to the Poisson-binomial process introduced in Section 1.1. A more comprehensive yet elementary presentation of some of these concepts (except those in Section 1.5), for the one-dimensional case and for traditional stochastic models (Markov chains, renewal, birth and death, queuing and Poisson processes), is found in any textbook on the subject, for instance in "Introduction to Stochastic Models" by R. Goodman [34].

1.4.1 Stationarity

There are various definitions of stationarity [Wiki] for point processes. The most common one is that the distribution of the point count $N(B)$ depends only on $\mu(B)$ (the length or area of B), but not on its location. The Poisson-binomial process is not stationary. Assuming $\lambda = 1$, if s is small enough, the point count distribution attached to (say) $B_1 = [0.3, 0.8]$ is different from that attached to $B_2 = [5.8, 6.3]$, despite both intervals having the same length. This is obvious if $s = 0$: in that case $N(B_1) = 0$, and $N(B_2) = 1$. However, if $B_1 = [a, b]$ and $B_2 = [a+k/\lambda, b+k/\lambda]$, then $N(B_1)$ and $N(B_2)$ have the same distribution, regardless of $k \in \mathbb{Z}$; see Theorem 4.1 for a related result. So, knowing the theoretical distribution of $N([x, x + 1/\lambda])$ for each $0 \leq x < 1/\lambda$ is enough to know the distribution of $N(B)$ on any interval B. Since λ is unknown when dealing with actual data, it must be estimated using techniques described in Section 3. This generalizes to two dimensions, with the interval $N([x, x+1/\lambda])$ replaced by the square $N([x, x+1/\lambda]) \times N([y, y+1/\lambda])$, with $0 \leq x, y < 1/\lambda$. Statistical testing is discussed in [55], also available online, here.

The interarrival times T face fewer non-stationarity issues, as evidenced by Theorem 4.3, Table 4, and Exercise 5. It should be favored over the point count $N(B)$, when assessing whether your data fit with a Poisson-binomial, or a Poisson point process model. In particular, it does not depend, for practical purposes, on the choice of X_0 in the definition of T in Section 1.2. The definition could be changed using (say) X_5, or any other X_k instead of X_0, with no impact on the theoretical distribution.

1.4.2 Ergodicity

This brings us to the concept of ergodicity. It is heavily used in the active field of dynamical systems: see [15, 19, 41] and my book [36] available here. I will cover dynamical systems in details, in my upcoming book on this topic. For Poisson-binomial point processes, ergodicity means that you can estimate a quantity in two different ways:

- using one very long simulation of the process (a large n in our case),
- or using many small realizations of the process (small n), and averaging the statistics obtained in each simulation

Ergodicity means that both strategies, at the limit, lead to to same value. This is best illustrated with the estimation of $\mathrm{E}[T]$, or its higher moments. The expectation of the interarrival times T is estimated, in most of my simulations, as the average distance between a point X_k, and its nearest neighbor to the right, denoted as X_k'. It is computed as an average of $X_k' - X_k$ over $k = -n, \ldots, n$ with $n = 3 \times 10^4$, on a single realization of the process. The same methodology is used in the source code provided in Section 6. Likewise, $\mathrm{E}[T^2]$ is estimated as the average $(X_k' - X_k)^2$ in the same way.

Table 4 is an exception. There I used 10^4 realizations of a same Poisson-binomial process. In each realization I computed, among others, $T_0 = X_0' - X_0$. This corresponds to the actual definition of T provided in Section 1.2. Then I averaged these T_0's over the 10^4 realizations to get an approximated value for T. It turns out that both methods lead to the same result. This is thanks to ergodicity, as far as T is concerned. I may as well have averaged $T_5 = X_5' - X_5$ over the 10^4 realizations, and end up with the same result for $\mathrm{E}[T]$. Note that not all processes are ergodic. The difference between stationarity and ergodicity is further explained here.

1.4.3 Independent Increments

A one dimensional point process is said to have independent increments or independent interarrival times if the point counts $N(B_1), N(B_2)$ for any two non-overlapping time intervals B_1, B_2 are independent. It is shown in some textbooks, for instance [71] (available online here), that the only stationary renewal process with independent increments is the stationary Poisson process. The proof is simple, and based on the fact that the only distribution having the memoryless property, is the exponential one. Another definition of independent increments [Wiki] is based on the independence of the successive interarrival times. If combined with "identically distributed", it allows you, for Poisson-binomial process, to choose any arbitrary k to define the interarrival times as the random variable $T = X_k' - X_k$, where X_k' is the closest neighbor point of X_k, to the right on the real line. These two definitions of "independent increments" are not equivalent, since the first one based on point count, is measured at arbitrary locations, while the second one, based on interarrival times, is (in one dimension) the interdistance between actual points of the process. The point count and interarrival times are related by the identity $P(T > y) = P[N(B_0) = 0]$, where $B_0 =]X_0, X_0 + y]$, see Section 4.2.

An off-the-beaten-path test of independence is discussed in Section 3.1.3, precisely to assess the assumption of independent increments, on simulated data. A related concept is the memoryless property [Wiki].

1.4.4 Homogeneity

An ordinary Poisson point process (the limit, as $s \to \infty$, of a Poisson-binomial process) is said to be homogeneous if the intensity λ does not depend on the location. In the case of the Poisson process, homogeneity is equivalent to stationarity. Even for non-homogenous Poisson processes, the point count $N(B_1)$ and $N(B_2)$, attached to two disjoint sets $B1, B_2$, are independently (though not identically) distributed. This is not the case for Poisson-binomial processes, not even for those that are homogeneous.

Poisson-binomial processes investigated so far are homogeneous. I discuss non-homogeneous cases in Sections 1.5.3, 1.5.4 and 2.1. A non-homogeneous Poisson-binomial process is one where the intensity λ depends on the index k attached to a point X_k.

1.5 Transforming and Combining Multiple Point Processes

I discuss here a few types of point process operations, including translation, rotation, superimposition, mixtures of point processes and marked point processes. Cluster point processes, a particular type of superimposed processes, are not introduced in this section: they are treated in detail in Section 2.1 and 3.4. Another type of operation called thinning (see [59], available online here), is not described in this textbook.

1.5.1 Marked Point Process

In one dimension, a marked point process is similar to a couple of paired time series. It has two components: the base process, modeled here as a Poisson-binomial process, and a "mark" (a random variable) attached to each point. In one dimension, the base process typically represents time occurrences of events, and marks represent some feature attached to each event. The definition easily generalizes to any dimension.

An example is the highest yearly flood occurrences for a particular river, over a long time period, say 200 years. Due to yearly recurrence, a Poisson-binomial process where the intensity is $\lambda = 1$ and the time unit is a year, is better suited than a standard Poisson process. The marks measure the intensity of each maximum yearly flood. Another, 3-D example, is the position of atoms in a crystal. The marks may represent the type of atom.

Formally, in one dimension, a marked point process is a (usually infinite) set of points (X_k, Y_k) with $k \in \mathbb{Z}$. The definition easily generalizes to higher dimensions. Typically, the Y_k's (the marks) are independently distributed, and independently distributed from the underlying process (X_k). The underlying process can be a Poisson-binomial process.

1.5.2 Rotation, Stretching, Translation and Standardization

In two dimensions, rotating a Poisson-binomial process is equivalent to rotating its underlying lattice attached to the index space. Rotating the points has the same effect as rotating the lattice locations, because F (the distribution attached to the points) belongs to a family of location-scale distributions [Wiki]. For instance, a $\pi/4$ rotation will turn the square lattice into a centered-square lattice [Wiki], but it won't change the main properties of the point process. Both processes, the original one and the rotated one, may be indistinguishable for all practical purposes unless the scaling factor s is small, creating model identifiability [Wiki] issues. For instance, the theoretical correlation between the point coordinates (X_h, Y_k) or the underlying lattice point coordinates $(h/\lambda, k/\lambda)$, measured on all points, remains equal to zero after rotation, because the number of points is infinite (this may not be the case if you observe points through a small window, because of boundary effects). Thus, a Poisson-binomial process has a point distribution invariant under rotations, on a macro-scale. This property is called anisotropy [Wiki]. On a micro-scale, a few changes occur though: for instance the two-dimensional version of Theorem 4.1 no longer applies, and the distance between the projection of two neighbor points on the X or Y axis, shrinks after the rotation.

Applying a translation to the points of the process, or to the underlying lattice points, results in a shifted point process. It becomes interesting when multiple shifted processes, with different translation vectors, are combined together as in Section 1.5.3. Theorem 4.1 may not apply to the shifted process, though it can easily be adapted to handle this situation. One of the problems is to retrieve the underlying lattice space of the shifted process. This is useful for model fitting purposes, as it is easier to compare two processes once they have been standardized (after removing translations and rescaling). Estimation techniques to identify the shift are discussed in Section 3.4.

By a standardized Poisson-binomial point process, I mean one in its canonical form, with intensity $\lambda = 1$, scaling factor $s = 1$, and free of shifts or rotations. Once two processes are standardized, it is easier to compare them, assess if they are Poisson-binomial, or perform various machine learning procedures on observed data, such as testing, computing confidence intervals, cross-validation, or model fitting. In some way, this is similar to transforming and detrending time series to make them more amenable to statistical inference. There is also some analogy between the period or quasi-period of a time series, and the inverse of the intensity λ of a Poisson-binomial process: in fact, $1/\lambda$ is the fixed increment between the underlying lattice points in the lattice space, and can be viewed as the period of the process.

Finally, a two dimensional process is said to be stretched if a different intensity is used for each coordinate for all the points of the process. It turns the underlying square lattice space into a rectangular lattice, and the homogeneous process into a non-homogeneous one, because the intensity varies locally. Observed data points can be standardized using the Mahalanobis transformation [Wiki], to remove stretching (so that variances are identical for both coordinates) and to decorrelate the two coordinates, when correlation is present.

1.5.3 Superimposition and Mixing

Here we are working with two-dimensional processes. When the points of m independent point processes with same distribution F and same index space $\mathbb{Z}^2$ are bundled together, we say that the processes are superimposed. These processes are no longer Poisson-binomial, see Exercise 14. Indeed, if the scaling factor s is small and $m > 1$ is not too small, they exhibit clustering around each lattice location in the lattice space. Also, the intensities or scaling factors of each individual point process may be different, and the resulting combined process may not be homogeneous. Superimposed point processes also called interlaced processes.

A mixture of m point processes, denoted as M, is defined as follows:

- We have m independent point processes $M_1, \dots, M_m$ with same distribution F and same index space $\mathbb{Z}^2$,
- The intensity and scaling factor attached to M_i are denoted respectively as λ_i and s_i $(i = 1, \dots, m)$,
- The points of M_i $(i = 1, \dots, m)$ are denoted as (X_{ih}, Y_{ik}); the index space consists of the (h, k)'s,
- The point (X_h, Y_k) of the mixture process M is equal to (X_{ih}, Y_{ik}) with probability $\pi_i > 0$, $i = 1, \dots, m$.

While mixing or superimposing Poisson-binomial processes seem like the same operation, which is true for stationary Poisson processes, in the case of Poisson-binomial processes, these are distinct operations resulting in significant differences when the scaling factors are very small (see Exercise 18). The difference is most striking when $s = 0$. In particular, superimposed processes are less random than mixtures. This is due to the discrete nature of the underlying lattice space. However, with larger scaling factors, the behavior of mixed and superimposed processes tend to be similar.

Several of the concepts discussed in Section 1.5 are illustrated in Figure 2, representing a realization of m superimposed shifted stretched Poisson-binomial processes, called m-interlacing. For each individual process M_i, $i = 1, \dots, m$, the distribution attached to the point (X_{ih}, X_{ik}) (with $h, k \in \mathbb{Z}$) is

$$P(X_{ih} < x, Y_{ik} < y) = F\Big(\frac{x - \mu_i - h/\lambda}{s}\Big) F\Big(\frac{y - \mu_i' - k/\lambda'}{s}\Big), \quad i = 1, \dots, m$$

This generalizes Formula (2). The parameters used for the model pictured in Figure 2 are:

- Number of superimposed processes: $m = 4$; each one displayed with a different color,
- Color: red for M_1, blue for M_2, orange for M_3, black for M_4,
- scaling factor: $s = 0$ (left plot) and $s = 5$ (right plot),
- Intensity: $\lambda = 1/3$ (X-axis) and $\lambda' = \sqrt{3}/3$ (Y-axis),
- Shift vector, X-coordinate: $\mu_1 = 0, \mu_2 = 1/2, \mu_3 = 2, \mu_4 = 3/2$,
- Shift vector, Y-coordinate: $\mu_1' = 0, \mu_2' = \sqrt{3}/2, \mu_3' = 0, \mu_4' = \sqrt{3}/2$,
- F distribution: standard centered logistic with zero mean and variance $\pi^2/3$.

For simulation purposes, the points (X_{ih}, Y_{ik}) of the i-th process M_i $(i = 1, \dots, m)$, are generated as follows:

$$X_{ih} = \mu_i + \frac{h}{\lambda} + s \cdot \log\Big(\frac{U_{ih}}{1 - U_{ih}}\Big) \tag{8}$$

$$Y_{ik} = \mu_i' + \frac{k}{\lambda'} + s \cdot \log\Big(\frac{U_{ik}}{1 - U_{ik}}\Big) \tag{9}$$

where U_{ij} are uniformly and independently distributed on $[0, 1]$ and $-n \leq h, k \leq n$. I chose $n = 25$ in the simulation – a window much larger than that of Figure 2 – to avoid boundary effects in the picture. The boundary effect is sometimes called edge effect. The unobserved data points outside the window of observations, are referred to as censored data [Wiki]. Of course, in my simulations their locations and features (such as which process they belong to) are known by design. But in a real data set, they are truly missing or unobservable, and statistical inference must be adjusted accordingly [23]. See also Section 3.5.

I discuss Figure 2 in Section 1.5.4. A simple introduction to mixtures of ordinary Poisson processes is found on the Memming blog, here. In Section 3.4, I discuss statistical inference: detecting whether a realization of a point process is Poisson or not, and detecting the number of superimposed processes (similar to estimating the number of clusters in a cluster process, or the number of components in a mixture model). In Section 3.4.4, I introduce a black-box version of the elbow rule to detect the number of clusters, of mixture components, or the number of superimposed processes.

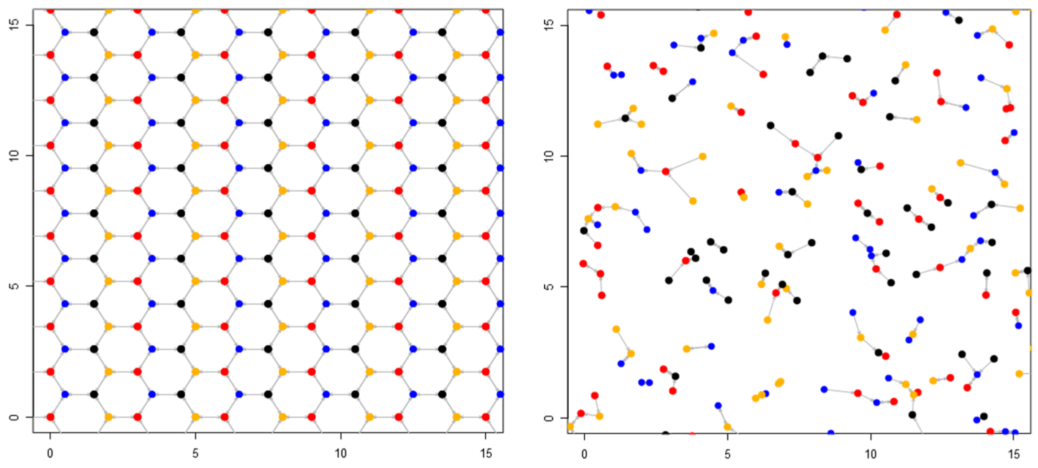

Figure 2: Four superimposed Poisson-binomial processes: $s = 0$ (left), $s = 5$ (right)

1.5.4 Hexagonal Lattice, Nearest Neighbors

Here I dive into the details of the processes discussed in Section 1.5.3. I also discuss Figure 2. The source code to produce Figure 2 is discussed in Sections 6.4 (nearest neighbor graph) and 6.7 (visualizations). Some elements of graph theory are discussed here, as well as visualization techniques.

Surprisingly, it is possible to produce a point process with a regular hexagonal lattice space using simple operations on a small number ($m = 4$) of square lattices: superimposition, stretching, and shifting. A stretched lattice is a square lattice turned into a rectangular lattice, by applying a multiplication factor to the X and/or Y coordinates. A shifted lattice is a lattice where the grid points have been shifted via a translation.

Each point of the process almost surely (with probability one) has exactly one nearest neighbor. However, when the scaling factor s is zero, this is no longer true. On the left plot in Figure 2, each point (also called vertex when $s = 0$) has exactly 3 nearest neighbors. This causes some challenges when plotting the case $s = 0$. The case $s > 0$ is easier to plot, using arrows pointing from any point to its unique nearest neighbor. I produced the arrows in question with the `arrow` function in R, see source code in Section 6.7, and online documentation here. A bidirectional arrow between points A and B means that B is a nearest neighbor of A, and A is a nearest neighbor of B. All arrows on the left plot in Figure 2 are bidirectional. Boundary effects are easily noticeable, as some arrows point to nearest neighbors outside the window. Four colors are used for the points, corresponding to the 4 shifted stretched Poisson-binomial processes used to generate the hexagon-based process. The color indicates which of these 4 process, a point is attached to.

The source code in Section 6.4 handles points with multiple nearest neighbors. It produces a list of all points with their nearest neighbors, using a hash table. A point with 3 nearest neighbors has 3 entries in that list: one for each nearest neighbor. A group of points that are all connected by arrows, is called a connected component [Wiki]. A path from a point of a connected component to another point of the same connected component, following arrows while ignoring their direction, is called a path in graph theory.

In my definition of connected component, the direction of the arrow does not matter: the underlying graph is considered undirected [Wiki]. An interesting problem is to study the size distribution, that is, the number of points per connected component, especially for standard Poisson processes. See Exercise 20. In graph theory, a point is called a vertex or node, and an arrow is called an edge. More about nearest neighbors is discussed in Exercises 18 and 19.

Finally, if you look at Figure 2, the left plot seems to have more points than the right plot. But they actually have roughly the same number of points. The plot on the right seems to be more sparse, because there are large areas with no points. But to compensate, there are areas where several points are in close proximity.

2 Applications

Applications of Poisson-binomial point processes (also called perturbed lattices point processes) are numerous. In particular, they are widely used in cellular and sensor network modeling and optimization. It also has applications in physics and crystal structures: see Figure 5 featuring man-made marble countertops. I provide many references in Section 2.1.

Here I focus on two-dimensional processes, to model lattice-based clustering. It is different from traditional clustering in two ways: clustering takes place around the vertices of the lattice space, and the number of clusters is infinite (one per vertex). This concept is visualized in Figures 15 and 16, showing representations of these cluster processes.

The processes in Section 2.1 are different from the mixtures or superimposed Poisson-binomial processes on shifted rectangular lattices, discussed in Sections 1.5.4 and 3.4. The latter can produce clustering on hexagonal lattice spaces, as pictured in Figure 2, with applications to cellular networks [Wiki]. See also an application to number theory (sums of squares) in my article "Bernoulli Lattice Models – Connection to Poisson Processes", available here. Instead, the cluster processes discussed here are based on square lattices and radial densities.

In Section 2.1, I introduce radial processes (called child processes) to model the cluster structure. The underlying distribution F attached to the points (X_h, Y_k) of the base process (called parent process) is the logistic one. In Section 2.1.1, I discuss a new type of generalized logistic distribution, which is easy to handle for simulation purposes, or to find its CDF (cumulative distribution function) and quantile function.

In Section 2.2, I focus on the hidden or inverse model (in short, the unobserved lattice). It leads to infinite, slightly random permutations. The final section deals with what the Poisson-binomial process was first designed for: randomizing mathematical series to transform them into random functions. The purpose is to study the effect of small random perturbations. Here it is applied to the famous Riemann zeta function. It leads to a new type of clusters called *sinks*, and awkward 2D Brownian motions with a very strong, unusual cluster structure, and beautiful data animations (see Section 2.4).

Along the lines, I prove Theorem 2.1, related to Le Cam's inequality. It is a fundamental result about the convergence of the Poisson-binomial distribution, to the Poisson distribution.

2.1 Modeling Cluster Systems in Two Dimensions

There are various ways to create points scattered around a center. When multiple centers are involved, we get a cluster structure. The point process consisting of the centers is called the parent process, while the point distribution around each center, is called the child process. So we are dealing with a two-layer, or hierarchical structure, referred to as a cluster point process. Besides clustering, many other types of point process operations [Wiki] are possible when combining two processes, such as thinning or superimposition. Typical examples of cluster point processes include Neyma-Scott (see here) and Matérn (see here).

Useful references include Baddeley's textbook "Spatial Point Processes and their Applications" [4] available online here, Sigman's course material (Columbia University) on one-dimensional renewal processes for beginners, entitled "Notes on the Poisson Process" [71], available online here, Last and Kenrose's book "Lectures on the Poisson Process" [52], and Cressie's comprehensive 900-page book "Statistics for Spatial Data" [16]. Cluster point processes are part of a larger field known as spatial statistics, encompassing other techniques such as geostatistics, kriging and tessellations. For lattice-based processes known as perturbed lattice point processes, more closely related to the theme of this textbook (lattice processes), and also more recent with applications to cellular networks, see the following references:

- "On Comparison of Clustering Properties of Point Processes" [12]. Online PDF here.
- "Clustering and percolation of point processes" [11]. Online version here.
- "Clustering comparison of point processes, applications to random geometric models" [13]. Online version here.
- "Stochastic Geometry-Based Tools for Spatial Modeling and Planning of Future Cellular Networks" [51]. Online version here.
- "Hyperuniform and rigid stable matchings" [54]. Online PDF here. Short presentation available here.
- "Rigidity and tolerance for perturbed lattices" [68]. Online version here.
- "Cluster analysis of spatial point patterns: posterior distribution of parents inferred from offspring" [66].
- "Recovering the lattice from its random perturbations" [79]. Online version here.
- "Geometry and Topology of the Boolean Model on a Stationary Point Processes" [81]. Online version here.
- "On distances between point patterns and their applications" [56]. Online version here.

More general references include two comprehensive volumes on point process theory by Daley and Vere-Jones [20, 21], a chapter by Johnson [45] (available online here or here), books by Møller and Waagepetersen, focusing on statistical inference for spatial processes [60, 61], and "Point Pattern Analysis: Nearest Neighbor Statistics" by Anselin [3] focusing on point inhibition/aggregation metrics, available here. See also [58] by Møller, available online here, and "Limit Theorems for Network Dependent Random Variables" [48], available online here.

Here, I use a two-dimensional Poisson-binomial process as the parent process to generate the centers of the cluster process (the child process). The child process, around each center, has a radial distribution. There are different ways to simulate radial processes; the most popular method uses a bivariate Gaussian distribution for the child process. Poisson point processes with non-homogeneous radial intensities are discussed in my article "Estimation of the Intensity of a Poisson Point Process by Means of Nearest Neighbor Distances" [35], freely available online here.

Remark: By non-homogeneous intensity, I mean that the intensity λ depends on the location, as opposed to a stationary Poisson process where λ is constant. Estimating the intensity function of such a process is equivalent to a density estimation problem, using kernel density estimators [Wiki].

To simulate radial distributions (also called radial intensities in this case), I use a generalized logistic distribution instead of the Gaussian one, for the child process. The generalized logistic distribution has nice features: easy to simulate, easy to compute the CDF, and it has many parameters, offering a lot of flexibility for the shape of the density. The peculiarity of the Poisson-binomial process offers two options:

- Classic option: Child processes are centered around the points of the parent process, with exactly one child process per point.
- Ad-hoc option: Child processes are centered around the bivariate lattice locations $(h/\lambda, k/\lambda)$, with exactly one child process per location, and $h, k \in \mathbb{Z}$.

In the latter case, if s is small, the child process attached to the index (h, k) has its points distributed around (X_h, X_k) – a point of the parent process – thus it won't be much different from the classic option. This is because if s is small, then $(h/\lambda, k/\lambda)$ is close to (X_h, X_k) on average. It becomes more interesting when s is neither too small nor too large.

In my simulations, I used a random number of points (up to 15) for the child process, and the parameter λ is set to one. I used a generalized logistic distribution for the radial distribution.

2.1.1 Generalized Logistic Distribution

In two dimensions, each point (X', Y') of a child process attached to a center (X, Y) of the parent process, is generated as follows, using the code in Section 6.3.

$$X' = X + \log\Big(\frac{U}{1-U}\Big)\cos(2\pi V) \tag{10}$$

$$Y' = Y + \log\Big(\frac{U}{1-U}\Big)\sin(2\pi V) \tag{11}$$

Here U and V are independent uniform deviates on $[0, 1]$. Let $Q = Q(U) = \log\frac{U}{1-U}$. It has a logistic distribution, centered at $\mu = 0$ and with scaling parameter $\rho = 1$.

I now introduce a generalized logistic distribution sharing the same features: very easy to sample, with a simple CDF, but this time with 5 parameters rather than 2: $\mu \in \mathbb{R}$ and $\alpha, \beta, \rho, \tau > 0$. The location parameter is μ, and the scaling parameter is ρ. The general form of its quantile function is given by

$$Q(u) = \mu + \rho\Big[\log\Big(\frac{\tau u^{1/\beta}}{1 - u^{1/\beta}}\Big)\Big]^{1/\alpha}, \quad 0 < u < 1. \tag{12}$$

The standard logistic distribution corresponds to $\alpha = \beta = \tau = 1$. In the general form, $\alpha = p/q$ where p, q are strictly positive co-prime odd integers, to avoid problems with negative logarithms. The function Q is known as the quantile function [Wiki]. It is the inverse of the CDF function $P(Z < z)$. The CDF is given by:

$$P(Z < z) = \Big[1 + \tau\exp\Big(-\Big(\frac{z-\mu}{\rho}\Big)^{\alpha}\Big)\Big]^{-\beta}. \tag{13}$$

Here ρ is the dispersion or scaling parameter. It was denoted as s when attached to the parent process, but one could choose a different value ($\rho \neq s$) for the child process, thus the introduction of the symbol ρ. Despite the 5 parameters, the CDF is rather simple. Also it is straightforward to generate deviates for this distribution, using (12) with uniform deviates on $[0, 1]$, for u. This technique is known as inverse transform sampling [Wiki].

In addition, the moments (and thus the variance) can be directly computed using the CDF. Whenever the density is symmetric around the origin (if $\mu = 0, \alpha = \beta = 1$ in our case) we have $\mathrm{E}[Z^r] = 0$ if r is an odd integer, and if r is even, we have

$$\mathrm{E}[Z^r] = 2r\int_0^\infty x^{r-1}P(Z > z)dz. \tag{14}$$

See Exercise 22 in Section 5 for a proof of this identity. An alternative formula to compute the moments is provided by the quantile function, see theorem 4.9:

$$E[Z^r] = \int_0^1 Q^r(u)du \tag{15}$$

Formulas (14) and (15) may be used to solve some integration problems. For instance, if a closed form can be found for (15), then the integral in (14) has the same value. I just mention three results here; more details are found in Exercise 3 in Section 5.

- If $\alpha = 1, \beta = 1/6$, then $\mathrm{E}[Z] = \mu + \rho \log \tau - \frac{\rho}{2}(\sqrt{3}\pi + 4\log 2 + 3\log 3)$ and the distribution is typically not symmetric.
- If $\alpha = 1$ and $\tau = e^{1/\beta}$, then $\beta^{-1}\mathrm{E}[(Z-\mu)/\rho] \to \pi^2/6$ as $\beta \to 0$. This is a consequence of (41). However, the limiting distribution has zero expectation. See Exercise 4 in Section 5 for details.
- If $\alpha = \beta = 1$, then $\mathrm{E}[(Z-\mu)/\rho] = \log \tau$ and $\mathrm{Var}[Z/\rho] = \frac{\pi^2}{3}$. The standard logistic distribution corresponds to $\tau = 1$.

Finally, the moment generating function [Wiki] (MGF) can easily be computed using the quantile function, as a direct application of the quantile theorem 4.9. If $\mu = 0$ and $\alpha = \rho = 1$, we have:

$$\begin{aligned}
\mathrm{E}[\exp(tZ)] &= \int_0^1 \exp\Big[t \log\Big(\frac{\tau u^{1/\beta}}{1-u^{1/\beta}}\Big)\Big] du \\
&= \tau^t \int_0^1 u^{t/\beta}(1-u^{1/\beta})^{-t} du \\
&= \beta\tau^t \int_0^1 v^{t+\beta-1}(1-v)^{-t} dv \\
&= \beta\tau^t B(\beta+t, 1-t), \qquad (16)
\end{aligned}$$

where B is the Beta function [Wiki]. Note that I made the change of variable $v = u^{1/\beta}$ when computing the integral. Unless $\alpha = \tau = 1$, it is clear from the moment generating function that this 5-parameter generalized logistic distribution is different from the 4-parameter one described in [Wiki]. Another generalization of the logistic distribution is the metalog distribution [Wiki].

Remark: If $\alpha = 1$, we face a model identifiability issue [Wiki]. This is because if $\tau_1 \exp(\mu_1/\rho_1) = \tau_2 \exp(\mu_2/\rho_2)$, the two CDF's are identical even if these two subsets of parameters are different. That is, it is impossible to separately estimate τ, μ and ρ. However, in practice, we use $\mu = 0$.

2.1.2 Illustrations

Figures 3 and 4 show two extreme cases of the cluster processes discussed at the beginning of Section 2.1. The parent process modeling the cluster centers, is Poisson-binomial. It is simulated with intensity $\lambda = 1$, using a uniform distribution for F. The scaling factor is $s = 0.2$ for Figure 3, and $s = 2$ for Figure 4. The left plot is a zoom-in. Around each center (marked with a blue cross in the picture), up to 15 points are radially distributed, creating the overall cluster structure. These points are the actual, observed points of the process, referred to as the child process. The distance between a point (X', Y') and its cluster center (X, Y) has a half-logistic distribution [Wiki]. The simulations are performed using Formulas (10) and (11).

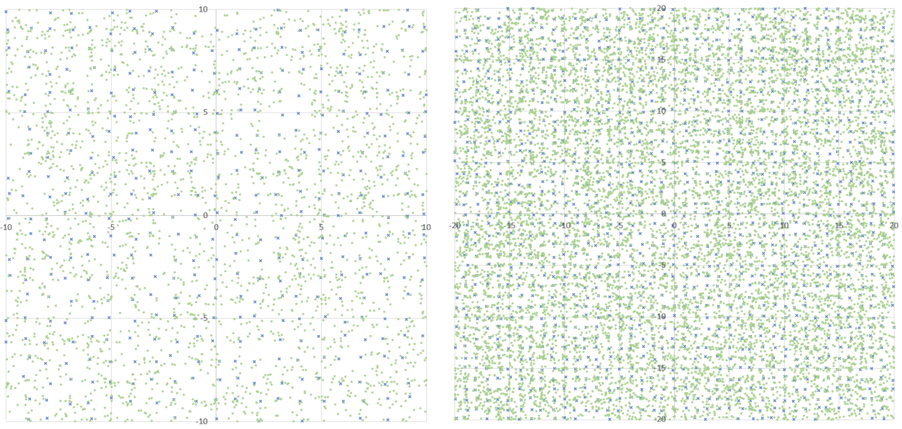

Figure 3: Radial cluster process ($s = 0.2, \lambda = 1$) with centers in blue; zoom in on the left

The contrast between Figures 3 and 4 is due to the choice of the scaling factor s. The value $s = 0.2$, close to zero, strongly reveals the underlying lattice structure. Here this effect is strong because of the choice of F (it has a very thin tail), and the relatively small variance of the distance between a point and its associated cluster center. It produces repulsion among neighbor points: we are dealing with a repulsive process, also called perturbed lattice point processes. When $s = 0$, all the randomness is gone: the state space is the lattice space. See left plot in Figure 2. Modeling applications include optimum distribution of sensors (for instance cell towers), crystal structures and bonding patterns of molecules in chemistry.

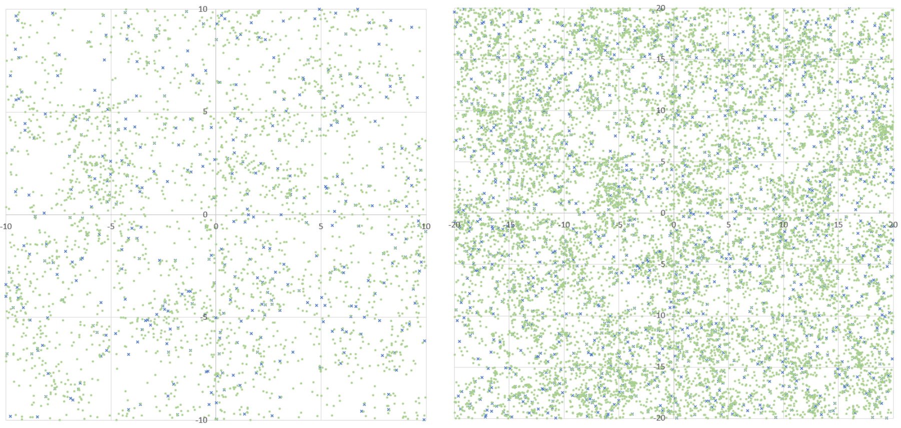

Figure 4: Radial cluster process ($s = 2, \lambda = 1$) with centers in blue; zoom in on the left

By contrast, $s = 2$ makes the cluster structure much more apparent. This time, there is attraction among neighbor points: we are dealing with an attractive process. It can model many types of structures, associated to human activities or natural phenomena, such as the distribution of galaxies in the universe. Figure 5 provides an example, related to the manufacture of kitchen countertops. I discuss other types of cluster patterns generated by Poisson-binomial processes, in Sections 2.4 and 3.4.

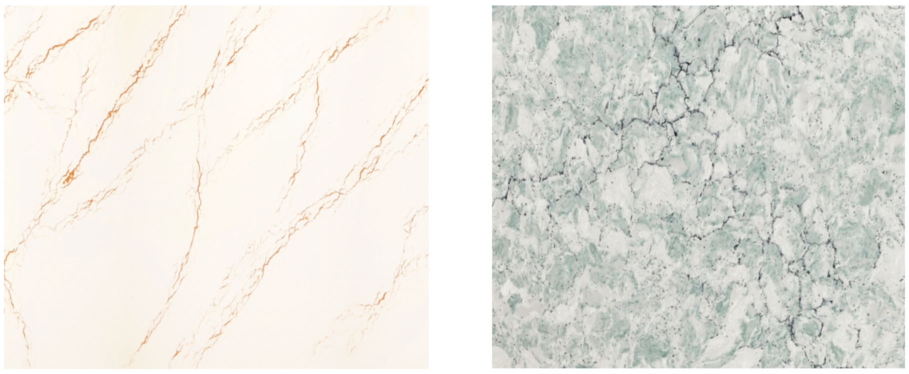

Figure 5: Manufactured marble lacking true lattice randomness (left)

Figure 5 shows luxury kitchen countertops called "Inverness bronze Cambria quartz", on the left. While the quality (and price) is far superior to all other products from the same company, the rendering of marble veins is not done properly. It looks man-made: not the kind of patterns you would find in real stones. The pattern is too regular, as if produced using a very small value of the scaling factor s. An easy fix is to used patterns generated by the cluster processes described here, incidentally called perturbed lattices. To increase randomness, increase s. It will improve the design. I am currently talking to the company, as I plan to buy these countertops. The picture on the right shows a more realistic rendering of randomness.

2.2 Infinite Random Permutations with Local Perturbations

The unobserved index k attached to any point X_k of the Poisson-binomial point process, gives rise to an interesting random process called the hidden process or index process, see Section 4.7. It can be used to generate infinite, locally random permutations (here in one dimension), using the following algorithm:

Algorithm: Generate a locally random permutation of order m

- **Step 1**: Generate a 1-D realization of a Poisson-binomial process with $2n + 1$ points $X_{-n}, \dots, X_n$.
  - Let $L(X_k) = k$, for $-n \leq k \leq n$. The function L is stored as an hash table [Wiki] in your source code; the keys of your hash table are the X_k's. In practice, no two X_h, X_k with $h \neq k$ have the same value $X_h = X_k$, so this collision problem won't arise.
- **Step 2**: Sort the $2n + 1$ points X_k, with $-n \leq k \leq n$.

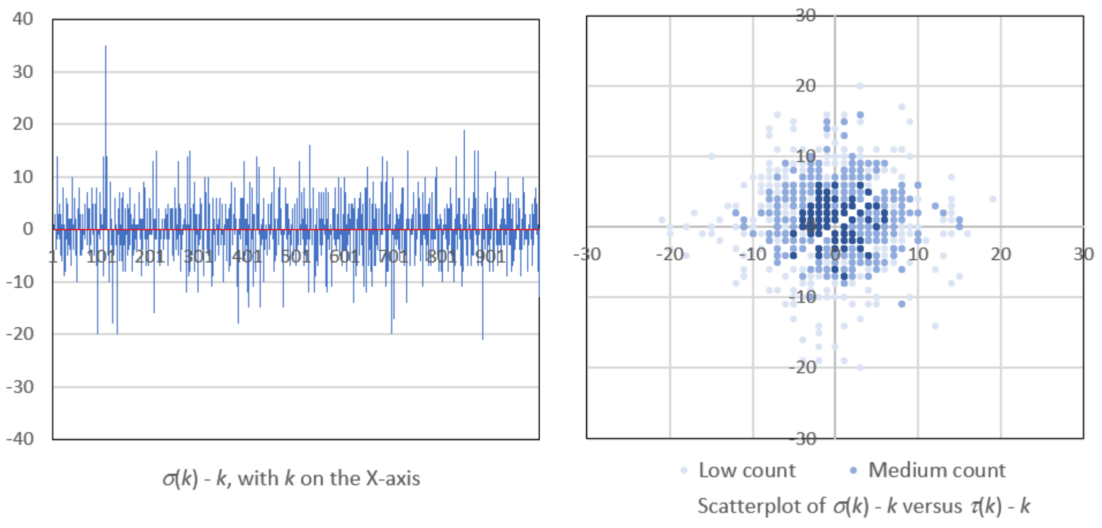

Figure 6: Locally random permutation σ; $\tau(k)$ is the index of X_k's closest neighbor to the right

- Denote as $X_{(k)}$ the k-th point after ordering.

- **Step 3**: Select m consecutive ordered points, say $X_{(1)}, \dots, X_{(m)}$ with m much smaller than n
  - Retrieve their original indices: $\sigma(k) = L(X_{(k)})$, $k = 1, \dots, m$
  - Set $\tau(k) = L(X_{(k+1)})$, $k = 1, \dots, m$ (so $X_{\tau(k)}$ is the closest point to $X_{\sigma(k)}$, to the right)

Now σ is a random permutation on $\{1, \dots, m\}$ [Wiki]. To produce the plots in Figure 6, I used $m = 10^3, n = 3 \times 10^4$ and a Poisson-binomial process with $\lambda = 1, s = 3$ and a logistic distribution for F. Since the theory is designed to produce infinite rather than finite permutations, boundary effects can take place. To minimize them, take both m and n large. The boundary effects, if present (for instance when using a thick tail distribution for F such as Cauchy, or when using a large s) will be most noticeable for $\sigma(k)$ when k is close to 1 or close to m.

These permutations can be used to model local reshuffling in a long series of events. Effects are mostly local, but tend to spread to longer distances on average, when s is large or F has a thick tail. For instance, in Figure 6, the biggest shift in absolute value is $\sigma(k) - k = 35$, occurring at $k = 108$ (see the peak on the left plot). However, peaks (or abysses) of arbitrary height will occur if m is large enough, unless you use a uniform distribution for F, or any distribution with a finite support domain.

The right plot in Figure 6 shows the joint empirical distribution (data-based as opposed to theoretical) of the discrepancies $\sigma(k) - k$ and $\tau(k) - k$ in the index space $\mathbb{Z} \times \mathbb{Z}$. Of course, since the index $\tau(k)$ points to the closest neighbor of $X_{\sigma(k)}$ to the right, that is, to $X_{\tau(k)}$, we have $\tau(k) \geq 1 + \sigma(k)$, which explains why the main diagonal is blank. Other than that, the plot shows independence, symmetry, and anisotropy (absence of directional trend in the scattering). It means that

- Given a point X_k, the index $\tau(k)$ of its nearest neighbor to the right is randomly distributed around k, according to some radial distribution,
- Given a point X_k, its order $\sigma(k)$ once the points are ordered, is randomly distributed around k, according to the same radial distribution,
- There is independence between the two.

Two metrics used to compare or describe these permutations are the average and maximum index discrepancy, measured as the average and maximum value of $|\sigma(k) - k|$ for $1 \leq k \leq m$. It gets larger as s increases. Another metric of interest, related to the entropy of the permutation [Wiki] [80], is the correlation between the integer numbers k and $\sigma(k) - k$, computed over $k = 1, \dots, m$. While the example featured in Figure 6 exhibits essentially a zero correlation, some other cases not reported here, exhibit a strong correlation. See also [2], available online on arXiv, here. For an elementary introduction to permutations, see [9].

2.3 Probabilistic Number Theory and Experimental Maths

Experimental mathematics is an active field where machine learning techniques are used to discover conjectures and patterns in number theory – for instance about the distribution of twin primes or the digits of π. References on this topic include [6, 10, 76] and my book [36], available here. Here I discuss two problems. The first one (Section 2.3.1) deals with the hypothetical count distribution of large factors in some intervals, in very large composite integers used in cryptography. It is obtained by simulations and heuristic arguments, and illustrates

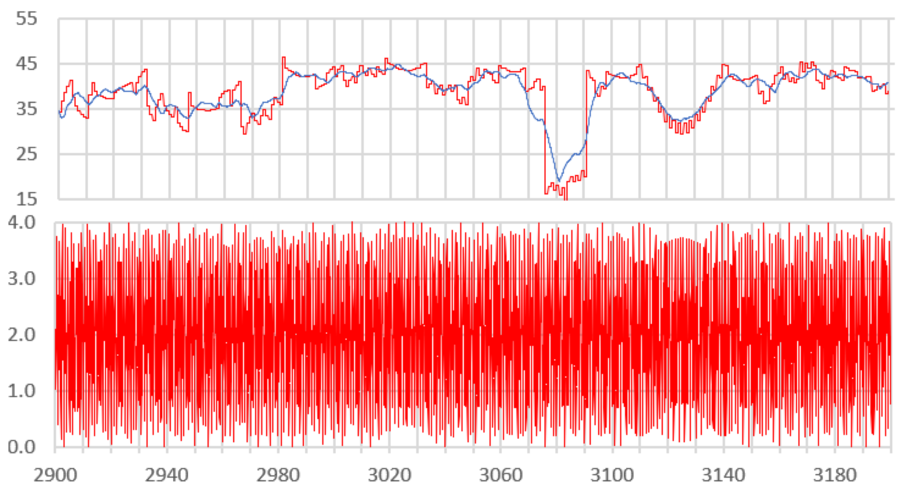

Figure 7: Chaotic function (bottom), and its transform (top) showing the global minimum

what probabilistic number theory is about. In the process, I prove a version of Le Cam's theorem: the fact that under certain circumstances, the Poisson-binomial distribution tends to a Poisson distribution.

The second problem (Section 2.3.2) deals with the Riemann zeta function ζ and the famous Riemann hypothesis (RH), featuring unusual, not well-known patterns It leads to heuristic arguments supporting RH. I then apply small perturbations to ζ (more specifically, to its sister function, the Dirichlet eta function η) using a Poisson-binomial process, to see when and if the patterns remain. The purpose is to check whether RH can be extended to a larger class of chaotic, random functions, unrelated to Dirichlet L-functions [Wiki]. Would such an extension be possible, it could offer new potential paths to proving RH. Unfortunately, while I exhibit such extensions, they only occur when the perturbations are incredibly small. RH has a \$1 million award attached to it and offered by the Clay Institute, see here.

2.3.1 Poisson Limit of the Poisson-binomial Distribution, with Applications

Historically, the problem discussed here has its origins in a new optimization technique to detect rare, hard-to-find global minima or roots of peculiar, highly irregular or discrete functions. An example is $g(b) = 2 - \cos(2\pi b) - \cos(2\pi a/b)$ with $a = 7919 \times 3083$, pictured in Figure 7. The technique uses a *divergent* fixed point algorithm [Wiki] that somehow leads to a solution. This material will be included in one of my upcoming textbooks. An overview of the (yet unpublished) technique can be found in "A New Machine Learning Optimization Technique - Part 1" available online, here. Initially, the application in mind was factoring numbers that are a product of two very large primes (the roots), typically used as encryption keys in cryptographic systems.

The bottom plot in Figure 7 represents the function $g(b)$ with $b \in [2900, 3100]$. It has a global minimum in that interval: $g(b) = 0$, occurring at $b = 3083$ (one of the only two factors of a). Note that g is differentiable and smooth everywhere. But its almost aperiodic oscillations have such a high frequency, that g looks chaotic to the naked eye, making the minimum invisible. Also, it is not possible to efficiently find the minimum using standard optimization algorithms. The top part of Figure 7 shows the same function, after discretization and magnification of the dip (caused by the minimum). Now the root-searching fixed-point algorithm can detect the minimum. It does that by emitting a strong signal when entering the deep valley, after a relatively small number of iterations. The blue curve is a smooth version of the discretized, step function in red [Wiki].

The remaining of this discussion focuses on a particular aspect of the problem in question. It features another case of convergence of the Poisson-binomial distribution to the Poisson distribution. The first case was a byproduct of the convergence of Poisson-binomial processes to standard Poisson processes (see Theorem 4.5). It implied that their point count distribution – a Poisson-binomial – must also converge to a Poisson distribution.

Description of the Problem

I now describe the number theoretic problem in probabilistic terms. The simulation consists of generating m independently distributed random positive integers $Z_1, \dots, Z_m$ with Z_k uniformly distributed on $\{0, 1, ..., n+k-1\}$. Here n is fixed but arbitrary, and we are interested in $n \to \infty$. Also, m is larger than n, with $m/n \to \alpha > 1$ as $n \to \infty$, typically with $\alpha = 2$. The probability that Z_k is zero is denoted as $p_k = P(Z_k = 0)$. The number of

Z_k's equal to zero, with $1 \leq k \leq m$, is denoted as $N(n, m)$ or simply N. In mathematical notations,

$$N = \sum_{k=1}^{m} \chi(Z_k = 0), \tag{17}$$

where χ is the indicator function [Wiki], equal to one if its argument is true, and to zero otherwise. Thus N, the counting random variable, has a Poisson-binomial distribution [Wiki] of parameters $p_1, \cdots, p_m$, similar to that discussed in Formula (4). The goal is to prove that when $n \to \infty$, the limiting distribution of N is Poisson with expectation $\log \alpha$. I then discuss the implications of this result, regarding the distribution of large factors in very large integers. The main result, Theorem 2.1, is a particular case of Le Cam's inequality [Wiki]; see also [73], available online here.

Theorem 2.1 *As $n \to \infty$ and $m/n \to \alpha > 1$, the discrete Poisson-binomial distribution of the counting random variable N defined by Formula (17), tends to a Poisson distribution of expectation $\log \alpha$.*

Proof

Clearly, $p_k = P(Z_k = 0) = 1/(n + k)$. Let

$$q_0 = \prod_{k=1}^{m} (1 - p_k) \text{ and } \mu = \sum_{k=1}^{m} \frac{p_k}{1 - p_k}.$$

We have, as in Formula (6) and (7), $P(N = 0) = q_0$ and $P(N = 1) = q_0\mu$. In Exercise 7, I prove that as $n \to \infty$, $P(N = k) = q_0\mu^k/k!$ for all positive integers k. Thus

$$\sum_{k=0}^{\infty} P(N = k) = q_0 \exp(\mu) = 1.$$

This corresponds to a Poisson distribution. It follows that $\mu = -\log q_0$. To complete the proof, I now show that $q_0 \to 1/\alpha$, as $n \to \infty$ and $m/n \to \alpha > 1$. We have

$$\begin{aligned}
\log q_0 = \log \prod_{k=1}^{m} (1 - p_k) &= \sum_{k=0}^{\infty} \log(1 - p_k) = -\sum_{k=1}^{m} p_k + O\Big(\frac{1}{n}\Big) \\
&= -\sum_{k=1}^{m} \frac{1}{n+k} + O\Big(\frac{1}{n}\Big) = -\sum_{k=1}^{m} \frac{1}{k} + \sum_{k=1}^{n} \frac{1}{k} + O\Big(\frac{1}{n}\Big) \\
&= -\log m + \log n + O\Big(\frac{1}{n}\Big) = -\log(m/n) + O\Big(\frac{1}{n}\Big) = -\log \alpha + O\Big(\frac{1}{n}\Big)
\end{aligned}$$

and thus $q_0 \to 1/\alpha$ as $n \to \infty$. ∎

Chance of Detecting Large Factors in Very Large Integers

In encryption systems, one usually works with very large integers a that only have two factors b and a/b, both large prime numbers, in order to make encryption keys secure. The reason is because factoring a very large integer that only has two large factors, is very difficult. The encryption keys are a function of the numbers b and a/b. The question is as follows: for a very large, "random" integer a (random in the sense that it could have any number of factors, including none), what are the chances that it has at least one factor b in the range $b \in [n, m]$? I assume here that n, m are very large too, with $m/n = \alpha > 1$ and $m < \sqrt{a}$. The answer, according to Theorem 2.1, is $P(N) > 0 = 1 - \exp[-\log \alpha] = 1 - 1/\alpha$. The expected number of factors, in that range, is $\mathrm{E}[N] = \log \alpha$. These are rough approximations, as it assumes randomness in the distribution of residues $a \bmod b$ when a is fixed and $b \in [n, m]$. This is explained in the next paragraph. By definition, $a \bmod b$ is the remainder in the integer division a/b [Wiki], also called *residue* in elementary number theory; mod is the modulo operator [Wiki] and b is called the *modulus*. In our example at the begining of Section 2.3.1, $a = 7919 \times 3083$. I used the interval $[n, m] = [2900, 3200]$ to find a factor of a. The chance to find one (namely, $b = 3083$) is approximately $1 - 2900/3200 \approx 69\%$ assuming you don't know what the factors are, a is random, and you picked up the interval $[n, m]$ at random.

To summarize, for a very large, arbitrary integer a, the number of factors in an interval $[n, m]$ with n large, $m < \sqrt{a}$, and $m/n \approx \alpha > 1$, approximately has a Poisson distribution of expectation $\log \alpha$. The explanation is as follows:

- $a \bmod n \in \{0, \ldots, n - 1\}$ is random,
- $a \bmod (n + 1) \in \{0, \ldots, n\}$ is random,

- $a \bmod (n+2) \in \{0, \dots, n+1\}$ is random,

  $\vdots$
- $a \bmod (m-1) \in \{0, \dots, m-2\}$ is random,
- $a \bmod m \in \{0, \dots, m-1\}$ is random,

and the above $m-n+1$ residues are mutually independent to some extent. The integer a has a factor in $[n, m]$ if and only if at least one of the above residues is zero. Thus Theorem 2.1 applies, at least approximately.

Integer Sequences with High Density of Primes

The Poisson-binomial distribution, including its Poisson approximation, can also be used in this context. The problem was originally posted here, and the purpose was to find fast-growing integer sequences with a very high density of primes.

The probability that a large integer x is prime is about $1/\log x$, a consequence of the prime number theorem [Wiki]. Let $x_1, x_2, \dots$ be a strictly increasing sequence of positive integers, and N denote the number of primes among $x_n, x_{n+1}, \dots, x_{n+m}$ for some large n. Assuming the sequence is independently and congruentially equidistributed, then N has a Poisson-binomial distribution of parameters $p_n, \dots, p_{n+m}$, with $p_k = 1/\log x_k$. It is unimportant to know the exact definition of congruential equidistribution. Roughly speaking, it means that the joint empirical distribution of residues across the x_k's, is asymptotically undistiguishable from that of a sequence of random integers. Thus a sequence where 60% of the terms are odd integers, do not qualify (that proportion should be 50%).

This result is used to assess whether a given sequence of integers is unusually rich, or poor, in primes. If it contains far more large primes than the expected value $p_n + \cdots + p_{n+m}$, then we are dealing with a very interesting, hard-to-find sequence, useful both in cryptographic applications and for its own sake. One can build confidence intervals for the number of such primes, based on the Poisson-binomial distribution under the assumption of independence and congruential equidistribution. A famous example of such a sequence (rich in prime numbers) is $x_k = k^2 - k + 41$ [Wiki].

If $n, m \to \infty$ and $p_n^2 + \cdots + p_{n+m}^2 \to 0$, then the distribution of N is well approximated by a Poisson distribution, thanks to Le Cam's theorem [Wiki].

2.3.2 Perturbed Version of the Riemann Hypothesis

When I first investigated Poisson-binomial processes, it was to study the behavior of some mathematical functions represented by a series. The idea was to add little random perturbations to the index k in the summation, in short, replacing k by X_k, turning the mathematical series into a random function, and see what happens. Here this idea is applied to the Riemann zeta function [Wiki]. The purpose is to empirically check whether the Riemann Hypothesis [Wiki] still holds under small perturbations, as non-trivial zeros of the Riemann zeta function ζ are very sensitive to little perturbations. Instead of working with $\zeta(z)$, I worked with its sister, the Dirichlet eta function $\eta(z)$ with $z = \sigma + it \in \mathbb{C}$ [Wiki]: it has the same non-trivial zeros in the critical strip $\frac{1}{2} < \sigma < 1$, and its series converges in the critical strip, unlike that of ζ. Its real and imaginary parts are respectively equal to

$$\Re[\eta(z)] = \Re[\eta(\sigma + it)] = -\sum_{k=1}^{\infty} (-1)^k \frac{\cos(t \log k)}{k^\sigma} \tag{18}$$

$$\Im[\eta(z)] = \Re[\eta(\sigma + it)] = -\sum_{k=1}^{\infty} (-1)^k \frac{\sin(t \log k)}{k^\sigma} \tag{19}$$

Note that i represents the imaginary unit, that is $i^2 = -1$. I investigated two cases: $\sigma = \frac{1}{2}$ and $\sigma = \frac{3}{4}$. I used a Poisson-binomial process with intensity $\lambda = 1$, scaling factor $s = 10^{-3}$ and a uniform F to generate the (X_k)'s and replace the index k by X_k in the two sums. I also replaced $(-1)^k$ by $\cos \pi k$. The randomized (perturbed) sums are

$$\Re[\eta_s(z)] = \Re[\eta_s(\sigma + it)] = -\sum_{k=1}^{\infty} \cos(\pi X_k) \cdot \frac{\cos(t \log X_k)}{X_k^\sigma} \tag{20}$$

$$\Im[\eta_s(z)] = \Re[\eta_s(\sigma + it)] = -\sum_{k=1}^{\infty} cos(\pi X_k) \cdot \frac{\cos(t \log X_k)}{X_k^\sigma} \tag{21}$$

Proving the convergence of the above (random) sums is not obvious. The notation η_s emphasizes the fact that the (X_k)'s have been created using the scaling factor s; if $s = 0$, then $X_k = k$ and $\eta_s = \eta$.

Figure 8 shows the orbits of $\eta_s(\sigma + it)$ in the complex plane, for fixed values of σ and s. The orbit consists of the points $P(t) = (\Re[\eta_s(\sigma + it)], \Im[\eta_s(\sigma + it)])$ with $0 < t < 200$, and t increasing by increments of 0.05. The plots are based on a single realization of the Poisson-binomial process. The sums converge very slowly, though there are ways to dramatically increase the convergence: for instance, Euler's transform [Wiki] or Borwein's method [Wiki]. I used 10^4 terms to approximate the infinite sums.

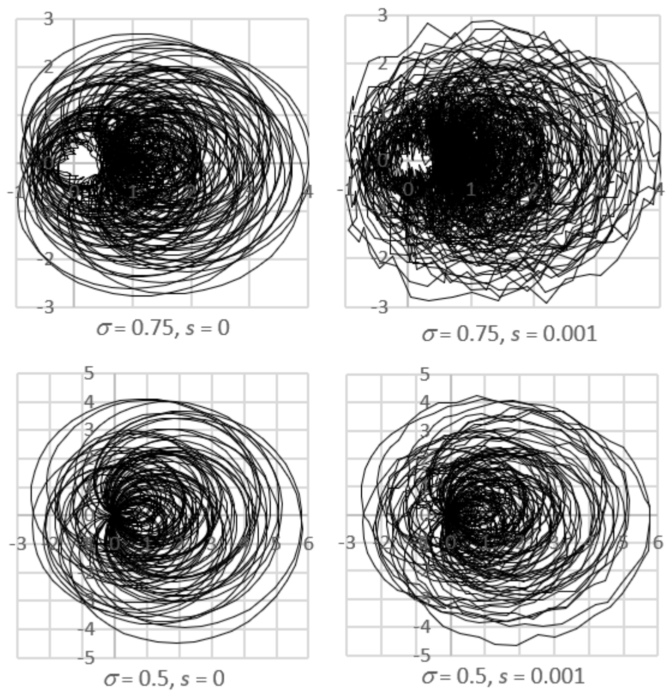

Figure 8: Orbit of η in the complex plane (left), perturbed by a Poisson-binomial process (right)

Let's look at the two plots on the left in Figure 8. A hole around the origin is noticeable when $\sigma = 0.75$. This suggests that η has no root with real part $\sigma = 0.75$, at least if $0 < t < 200$, as the orbit never crosses the origin, and indeed stays far away from it at all times. For larger t's the size of the hole may decrease, but with appropriate zooming, it may never shrink to an empty set. This is conjectured to be true for any $\sigma \in]\frac{1}{2}, 1[$ and any t; indeed, this constitutes the famous Riemann Hypothesis. To the contrary, if $\sigma = 0.5$, the orbit crosses the origin time and again, confirming the well-know fact that η has all its non-trivial zeros (infinitely many), on the critical line $\sigma = \frac{1}{2}$. This is the other part of the Riemann Hypothesis.

I noticed that the hole observed when $\sigma = 0.75$ shifts more and more to the left as σ decreases. Its size also decreases, to the point that when $\sigma = \frac{1}{2}$ (but not before), the hole has completely vanished, and its location has shifted to the origin. For $\sigma = 0.75$, it seems that there is a point on the X-axis, to the left-hand side of the hole but close to it, where the orbit goes through time and again, playing the same role as the origin does to $\sigma = \frac{1}{2}$. That special point, let's name it $h(\sigma)$, exists for any $\sigma \in [\frac{1}{2}, 1[$, and depends on σ. It moves to the right as σ increases. At least that's my conjecture, which is a generalization of the Riemann Hypothesis.

Let's now turn to the two plots on the right in Figure 8. I wanted to check if the above features were unique to the Riemann zeta function. If that is the case, it further explains why the Riemann Hypothesis is so hard to prove (or possibly unprovable), and why it constitutes to this day one of the most famous unsolved mathematical problems of all times. Indeed, there is very little leeway: only extremely small perturbations keep these features alive. For instance, using $s = 10^{-3}$ and a uniform F, that is, a microscopic perturbation, the orbits shown on the right are dramatically changed compared to their left counterparts. Key features seem to barely be preserved, and I suspect the hole, when $\sigma = 0.75$ no longer exists if you look at larger values of t: all that remains is a lower density of crossings where the hole used to be, compared to no crossing at all in the absence of perturbations ($s = 0$).

I will publish an eBook with considerably more details about the Riemann Hypothesis (and the twin prime conjecture) in the near future. The reason, I think, why such little perturbations have a dramatic effect, is because of the awkward chaotic convergence of the above series: see details with illustrations in Exercises 24 and 25, as well as here. The Riemann function gives rise to a number of interesting probability distributions, some related to dynamical systems, some defined on the real line, and some on the complex plane. This will be discussed in another upcoming book.

Remark: The conjecture that if $\sigma \in]1/2, 1[$, the hole never shrinks to a single point no matter how large t is (a conjecture weaker than the Riemann Hypothesis) must be interpreted as follows: it never shrinks to a point in any finite interval $[t, t + \tau]$. If you consider an infinite interval, this may not be true due to the universality of the Riemann zeta function [Wiki]. An approach to the Riemann hypothesis, featuring new developments, and not involving complex analysis, can be found in my article "Fascinating Facts About Complex Random Variables and the Riemann Hypothesis", here. For an introduction to the Riemann zeta function and Dirichlet series, see [44]. See also Section 2.4.1 in this textbook.

2.4 Videos: Fractal Supervised Classification and Riemann Hypothesis

This section combines many of the topics discussed in the textbook. The purpose is twofold: to teach you how to produce data videos, and to illustrate several of the topics covered throughout this textbook. Section 6.7 contains the source code, data sets and instructions to produce the videos. Here I focus on the statistical and mathematical methodology. To view one of the videos on YouTube, click on its thumbnail picture in Figure 9.

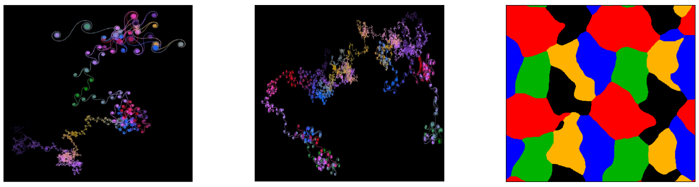
(a) Dirichlet 1 (b) Dirichlet 2 (c) clustering

Figure 9: Data animations – click on a picture to start a video

The two leftmost videos illustrate the beautiful, semi-chaotic convergence of the series attached to the Dirichlet eta function $\eta(z)$ [Wiki] in the complex plane. Details are in Section 2.4.1, including the connection to the famous Riemann Hypothesis [Wiki]. The rightmost video shows fractal supervised clustering performed in GPU (graphics processing unit), using image filtering techniques that act as a neural network. It is discussed in Section 2.4.2. For a short beginner introduction on how to produce these videos, read my article "Data Animation: Much Easier than you Think!", here.

2.4.1 Dirichlet Eta Function

Let $z = \sigma + it$ be a complex number, with σ the real part, and t the imaginary part. The Dirichlet eta function $\eta(z)$ provides an analytic continuation [Wiki] of the Riemann series $\zeta(z)$ in the complex plane ($z \in \mathbb{C}$). The two functions are defined as:

$$\zeta(z) = \sum_{k=1}^{\infty} \frac{1}{k^s}, \quad \sigma = \Re(z) > 1, \tag{22}$$

$$\eta(z) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k^s}, \quad \sigma = \Re(z) > 0. \tag{23}$$

Thus, the function ζ can be uniquely extended to $\sigma > 0$, using $\zeta(z) = (1 - 2^{1-z})^{-1}\eta(z)$, while preserving Formula (22) if $\sigma > 1$: the first series converges if and only if $\sigma > 1$, and the second one if and only if $\sigma > 0$. Both functions, after the analytic continuation of ζ, have the same zeroes in the critical strip $0 < \sigma < 1$. The famous Riemann Hypothesis [Wiki] claims that all the infinitely many zeroes in the critical strip occur at $\sigma = \frac{1}{2}$. This is one of the seven Millenium Problems, with a $1 million prize, see here. For another one, "P versus NP", see Exercise 21, about finding the maximum cliques of a nearest neighbor graph.

More than 10^{13} zeroes of ζ have been computed. The first two million are in Andrew Odlyzko's table, here. See the OEIS sequences A002410 and A058303. You can find zeroes with the free online version of Mathematica using the `FindRoot[]` and `Zeta[]` functions, here. For fast computation, several methods are available, for example the Odlyzko–Schönhage algorithm [Wiki]. The statistical properties are studied in Guilherme França and André LeClair [28] (available online here), in André LeClair in the context of random walks [53] (available online here) and in Peter J. Forrester and Anthony Mays in the context of random matrix theory [27] (available online here). I discuss recent developments about the Riemann Hypothesis in my article "Fascinating Facts About Complex Random Variables and the Riemann Hypothesis", here. See also my contributions on MathOverflow: "More mysteries about the zeros of the Riemann zeta function" (here) and "Normal numbers, Liouville function, and the Riemann Hypothesis" (here).

Connection to Poisson-binomial Processes

Rather than directly studying $\eta(z)$, I am interested in applying small perturbations to the summation index k (playing the role of a one-dimensional lattice) in Formula (23). In short, replacing k by X_k, $k = 1, 2$ and so on, where the (X_k)'s constitute a Poisson-binomial process. This turns $\eta(z)$ into a random function $\eta'(z)$ with real and imaginary parts defined respectively by Formulas (20) and (21). The question is this: does the Riemann Hypothesis also apply to the randomized version?

Unfortunately, the answer is negative, unless the scaling factor s in the underlying Poisson-binomial process is very close to zero: see Section 2.3.2. In short, if $s > 0$, $\eta'(z)$ – unlike $\eta(z)$ – may have zeroes in the critical strip $0 < \sigma < 1$, with $\sigma \neq \frac{1}{2}$. A positive answer would have provided some hope and a new path of attack. Another similar attempt, somewhat more promising, is discussed in my article "Deep visualizations to Help Solve Riemann's Conjecture", here. Again, σ is the real part of z. Note that if $s = 0$, then $\eta(z) = \eta'(z)$.

The videos in Figure 9 (on the left) show the successive partial sums of $\eta'(z)$ in the complex plane. The orbits in the video, depending on s and $z = \sigma + it$, show the chaotic convergence using 10,000 terms in the summation Formulas (20) and (21). If t is large (say $t = 10^5$), you usually need much more than 10,000 terms to reach the convergence zone. Also, I use a Weibull or Fréchet distribution of parameter γ, for the underlying Poisson-binomial process: see Formula (37). For standardization purposes discussed in the same section, the intensity is set to $\lambda = \Gamma(1 + \gamma)$.

The middle video in Figure 9 shows the convergence path of two orbits (corresponding to two different parameter sets) at the same time, to make comparisons easier. It would be interesting to use a zero of the Riemann zeta function for $z = \sigma + it$: for instance, $\sigma = \frac{1}{2}$ and $t \approx 14.134725$. The algorithm to produce the partial sums is in the `PB_inference.xlsx` spreadsheet, in the the `Video_Riemann` tab. The parameters σ, t, s, γ are in cells `B2:B5` for the first z, and `C2:C5` for the second one. For more details and source code, see Section 6.7.1.

The Story Told by the Videos

The video starts with a chaotic orbit that looks like a Brownian motion. The orbit then gets smoother, jumping from sink to sink until eventually entering a final sink and converging. When $s = 0$, the behavior is similar to that pictured in Figure 20. When $s > 0$ and $\gamma \neq 0$, the whole orbit looks like a Brownian motion. As s and γ get larger, the Brownian motion starts exhibiting a strong clustering structure, with well separated clusters called "sinks". This is visible in Figure 21. See the discussion accompanying these figures, for additional details about the sinks, and the Brownian versus clustered-Brownian behavior. My question "Is this a Brownian motion?", posted here on MathOverflow, brings more insights.

The cause of the sinks is well-known, and explained in Exercise 25, for the one-dimensional case. The orbits are very sensitive to small changes in the parameters, especially to tiny moves from the base model $s = 0$. Large values of t produce a large number of sinks; the behavior is radically different when t is close to zero. Values of σ between 0.1 and 0.6 produce similar patterns. Outside this range, the patterns are noticeably different.

The video featuring two orbits has this peculiarity: the orbit on the left, with $s = 0$, is non-Brownian; the one on the right with $s = 0.05$ and $\gamma = 0.005$ is slightly Brownian (barely, because s is still very close to zero, yet the curve is a bit less "curvy"). Despite the tiny difference in s, which makes you think that both orbits should converge to close locations, in reality the two orbits move in radically different directions from the very beginning: this is a typical feature of chaotic dynamical systems. In this case, it is caused by choosing a large value for t ($t \approx 5.56 \times 10^6$).

The observations (2D points) that generate the orbits, are realizations of a new, very rich class of point processes. Such point processes could have applications in specific contexts (possibly astronomy), as potential modeling tools. Identifying the sinks and counting their number can be done using unsupervised clustering techniques. One might even use the technique described in Section 3.4.3. Finally, the color harmony results from using harmonics, that is, cosine waves with specific periods: see Section 6.7.1 for explanations. The next step is to design a black-box algorithm for palette creation, and to automatically generate and add a soundtrack to the video, using related mathematical formulas that produce harmonic sounds. In short, AI-generated art!

2.4.2 Fractal Supervised Classification

The rightmost video in Figure 9 shows supervised clustering in action, from the first frame representing the training set with 4 groups, to the last one showing the cluster assignment of any future observation (an arbitrary point location in the state space). Based on image filtering techniques acting as a neural network, the video illustrates how machine learning algorithms are performed in GPU (graphics processing unit). GPU-based clustering [Wiki] is very fast, not only because it uses graphics processors and memory, but the algorithm itself has a computational complexity that beats (by a long shot) any traditional classifier. It does not require the computation of nearest neighbor distances.

The video medium also explains how the clustering is done, in better ways than any text description could do. You can view the video (also called data animation) on YouTube, here. The source code and instructions to help you create your own videos or replicate this one, is in Section 6.7.2. See Section 3.4.3 for a description of the underlying supervised clustering methodology.

I use the word "fractal" because the shape of the clusters, and their boundaries in particular, is arbitrary. The boundary may be as fractal-like as a shoreline. It also illustrates the concept of fuzzy clustering [Wiki]: towards the middle of the video, when the entire state space is eventually classified, constant cluster re-assignments are taking place along the cluster boundaries. A point, close to the fuzzy border between clusters A and B, is sometimes assigned to A in a given video frame, and may be assigned to B in the next one. By averaging cluster assignments over many frames, it is possible to compute the probability that the point belongs to A or B. Another question is whether the algorithm (the successive frames) converge or not. It depends on the parameters, and in this case, stochastic convergence is observed. In other words, despite boundaries changing all the time, their average location is almost constant, and the changes are small. Small portions of a cluster, embedded in another cluster, don't disappear over time.

Color Palette Optimization

Finally, the choice of the colors is not arbitrary. You want to use high contrast colors, so that the eye can easily distinguish between two clusters. To achieve this goal, I designed a palette optimization algorithm, especially useful when the number of clusters is large. Let m be the number of clusters. It works as follows.

- Step 1: Generate m random colors (R_i, G_i, B_i), $i = 1, \dots, m$, each one assigned to a cluster. The RGB vector represents the red, green and blue components; each coordinate is an integer between 0 and 255. Compute the minimum distance δ between any pair of colors.
- Step 2: Pick up one of the colors c and generate a random color c'. Compute the new minimum distance δ', assuming c is replaced by c'. If $\delta' > \delta$, replace c by c' otherwise don't make the change. Repeat Step 2 until the coloring of the clusters is visually good enough.

In step 1, in combination with using random colors, one can include prespecified colors such as red, blue, dark green and orange, as they constitute a good starting point. Interestingly, what the algorithm accomplishes is this: finding points (colors) in 3D, randomly distributed around lattice vertices in the RGB cube; the optimum lattice is the one maximizing distances between vertices. In short, I created a realization of a 3D Poisson-binomial point process, where the points are the colors! Two solutions achieving the optimum when $m = 4$ are the color sets {black, yellow, purple, cyan} and {white, red, green, blue}. For $m > 4$, see here

3 Statistical Inference, Machine Learning, and Simulations

This section covers a lot of material, extending far beyond Poisson-binomial processes. The main type of processes investigated here is the m-interlacing defined in Section 1.5.3, as opposed to the radial cluster processes studied in Section 2.1. An m-process is a superimposition of m shifted Poisson-binomial processes, well suited to model cluster structures. In Section 3.4.3, I discuss supervised and unsupervised clustering algorithms applied to simulated data generated by m-processes. The technique, similar to neural networks, relies on image filtering performed in the GPU (graphics processing unit). It leads to fractal supervised clustering, illustrated with data animations. I discuss how to automatically detect the number of clusters in Section 3.4.4.

Before getting there, I describe different methods to estimate the core parameters of these processes. First in one dimension in Section 3.2, then in two dimensions in Section 3.4.2. The methodology features a new test of independence (Section 3.1.3), model fitting via the empirical distribution, and dual confidence region in the context of minimum contrast estimation (Section 3.1.1). I show that the point count expectations are almost stationary but exhibit small periodic oscillations (Section 3.1.2) and that the increments (point counts across non-overlapping, adjacent intervals) are almost independent.

In many instances, Poisson-binomial processes exhibit patterns that are invisible to the naked eye. In Section 3.3, I show examples of such patterns. Then, I discuss model identifiability, and the need for statistical or machine learning techniques to unearth the invisible patterns. Boundary effects, their impact, and how to fix this problem, is discussed mainly in Section 3.5.

3.1 Model-free Tests and Confidence Regions

In 1979, Bradley Efron published his seminal article "Bootstrap Methods: Another Look at the Jackknife" [24], available online here. It marked the beginning of a new era in statistical science: the development of model-free, data driven techniques. Several chapters in my book "Statistics: New Foundations, Toolbox, and Machine Learning Recipes" [37] published in 2019 (available online here) deal with extensions and modern versions of this methodology. I follow the same footsteps here, first discussing the general principles, and then showing how it applies to estimating the intensity λ and scaling factor s of a Poisson-binomial process. As in Jesper Møller [58], my methodology is based on minimum contrast estimation: see slides 114-116 here or here. See also [18] for other examples of this method in the context of point process inference.

There are easier methods to estimate λ and s: I describe some of them in Section 3.2. However, the goal here is to provide a general framework that applies to any multivariate parameter. I chose the parameters λ, s as they are central to Poisson-binomial processes. By now, you should be familiar with them. They serve as a test to benchmark the methodology. Yet, the standard estimator of λ is slightly biased, and the method in this section provides an alternative to obtain unbiased estimates. It assumes that boundary effect are properly handled. I describe how to deal with them in Section 3.1.2.

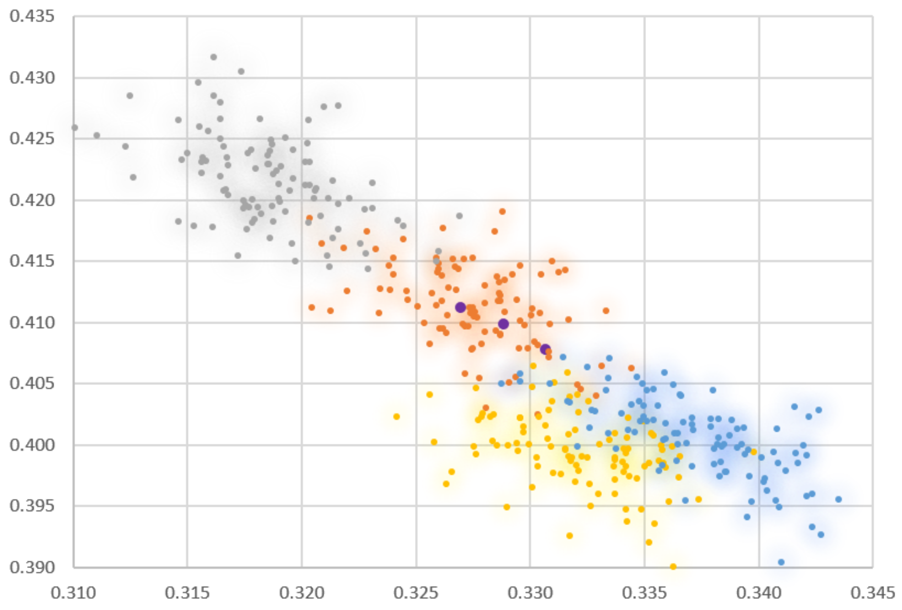

Figure 10: Minimum contrast estimation for (λ, s)

The idea behind minimum contrast estimation is to use proxy statistics as substitutes for the parameter estimators. It makes sense here as it is not clear what combination of variables represents s.

3.1.1 Methodology and Example

The observations consist of $2n+1$ points X_k $(k = -n, \dots, n)$ realization of a one-dimensional Poisson-binomial process of intensity λ and scaling factor s, obtained by simulation. I chose a logistic F in the simulation. Unless F has an unusually thick or thin tail, it has little impact on the point distribution. Let

$$R = \frac{1}{2n+1}\Big[\max_{|k|\leq n} X_k - \min_{|k|\leq n} X_k\Big], \tag{24}$$

$$B_k = \Big[\frac{k}{R}, \frac{k+1}{R}\Big[, \quad k = -n, \dots, n-1 \tag{25}$$

and

$$p = \frac{1}{2n}\sum_{k=-n}^{n-1} \chi[N(B_k) = 0], \quad q = \frac{1}{2n}\sum_{k=-n}^{n-1} \chi[N(B_k) = 1], \tag{26}$$

where χ is the indicator function [Wiki] and $N(B_k)$ is the number of points in B_k. If there is a one-to-one mapping between (λ, s) and (p, q), then one can easily compute (p, q) using Formula (26) applied to the observed data, and then retrieve (λ, s) via the inverse mapping. It is even possible to build 2D confidence regions for the bivariate parameter (λ, s). That's it!

I now explain how to implement this generic method to our example. I also address some of the challenges. First, the problem is to find good proxy statistics for the model parameters λ, s. I picked up p and q because it leads to an easy implementation in Excel. However, interarrival times (their mean and variance) are better, requiring smaller samples to achieve the same level of accuracy. Next, we are not sure if the mapping in question is one-to-one.

The scatterplot in Figure (10) illustrates the method. The X axis represents p, and the Y axis represents q. There are two main features:

- **Observed data**. The purple dots correspond to values of (p, q) derived from the observations, and computed with Formula (26). I tested three sets of observations (thus the three purple dots), each with 20,001 points (that is, $n = 10{,}000$).
- **Theoretical model**. The four overlapping clusters show the distribution of (p, q) for four different values of (λ, s). Each cluster – identified by its color – has 100 points corresponding to 100 simulations. Each simulation within a same cluster uses the same hand-picked (λ, s). Also, each simulation consists of $2n+1$ data points, to match the number of observations. The purpose of these simulations is to find the inverse mapping via numerical approximations. Four colors is just a small beginning. In Table 2, each cluster is summarized by two statistics: its computed center in the (p, q)–space, associated to the hand-picked parameter vector (λ, s).

Point Estimates

Let us focus on the rightmost purple dot in Figure 10, corresponding to one of the three observations sets. Its coordinates vector is denoted as (p_0, q_0). The (p, q)–space is called the proxy space. In this case, it is equal to $[0, 1] \times [0, 1]$. If the proxy spaced contained only the four points (p, q) listed in Table 2, the estimated value (λ_0, s_0) of (λ, s) would be the center of the orange cluster. That is, $(\lambda_0, s_0) = (1.4, 0.6)$ because $(0.3275, 0.4113)$ is the closest cluster center to the purple dot (p_0, q_0) in the proxy space.

But let's imagine that I hand-picked 10^5 vectors (λ, s) instead of four, thus generating 10^5 cluster centers and a very large Table 2 with 10^5 entries. Then again, the best estimator of (λ, s) would still be the one obtained by minimizing the distance between the purple dot (p_0, q_0) computed on the observations, and the 10^5 cluster centers. In practice, the hand-picking is automated (computerized) and leads to a black-box implementation of the estimation procedure.

Cluster	(λ, s)	(p, q)
Orange	$(1.4, 0.6)$	$(0.3275, 0.4113)$
Gray	$(1.4, 0.5)$	$(0.3186, 0.4216)$
Yellow	$(1.6, 0.7)$	$(0.3321, 0.3995)$
Blue	$(1.8, 0.6)$	$(0.3371, 0.4007)$

Table 2: Extract of the mapping table used to recover (λ, s) from (p, q)

Thanks to the law of large numbers [Wiki], the cluster centers quickly converge to their theoretical value as n increases. The cluster centers (p, q) in Table 2 can be computed as a function of (λ, s) using a mathematical formula. It facilitates the construction of the inverse mapping, avoiding tedious simulations: see Section 3.1.2.

Confidence Regions

Again, for the sake of illustration, let us focus on the rightmost purple dot (p_0, q_0). Imagine that contour lines are drawn around each cluster center. A contour line of level γ $(0 \le \gamma \le 1)$ is a closed curve (say an ellipse) oriented in the same direction as the cluster in question, and centered at the cluster center. Its interior covers a proportion γ of the points of the cluster, in the proxy space. In this case, the contour line of level γ, around the cluster center (p, q) is obtained as follows.

First define

$$H_n(x, y, p, q) = \frac{2n}{1-\rho_{p,q}^2} \cdot \Big[\Big(\frac{x-p}{\sigma_p}\Big)^2 - 2\rho_{p,q}\Big(\frac{x-p}{\sigma_p}\Big)\Big(\frac{y-q}{\sigma_q}\Big) + \Big(\frac{y-q}{\sigma_q}\Big)^2\Big], \tag{27}$$

with

$$\sigma_p = \sqrt{p(1-p)}, \quad \sigma_q = \sqrt{q(1-q)}, \quad \rho_{p,q} = -\frac{pq}{\sqrt{pq(1-p)(1-q)}}. \tag{28}$$

Then the contour line is the set of points $(x, y) \in [0, 1] \times [0, 1]$ satisfying $H_n(x, y, p, q) = G_\gamma$. Here G_γ is a quantile of some Hotelling distribution [Wiki]. I included a table of the G_γ function, obtained by simulations, in my spreadsheet (see next section); it is also pictured on the right plot in Figure 11.

This classic asymptotic result is a consequence of the central limit theorem, see here. For detailed explanations, see Exercise 27. Note that G_γ does not depend on n, p or q. At least not asymptotically.

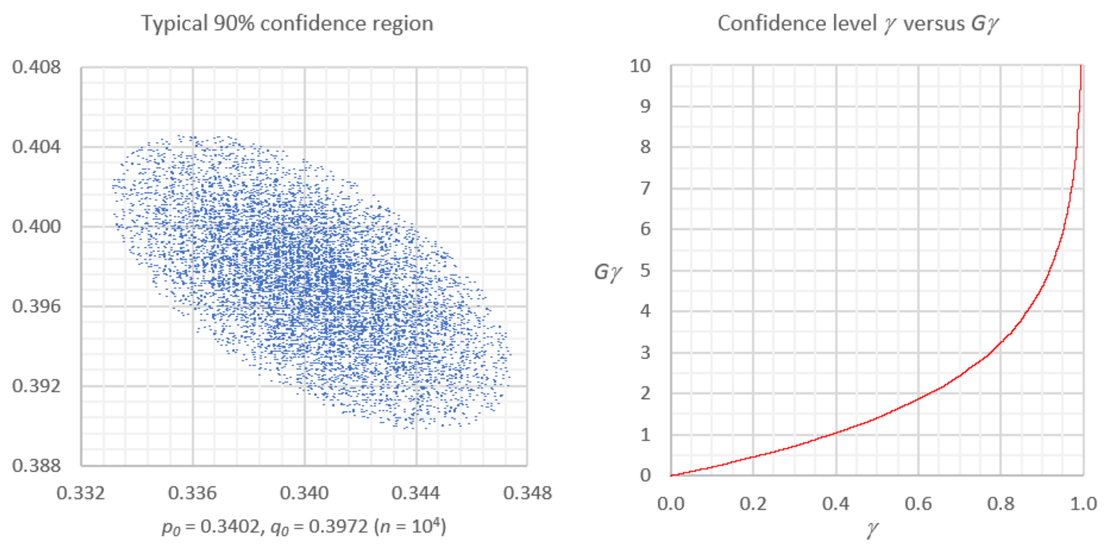

Figure 11: Confidence region for (p, q) – Hotelling's quantile function on the right

We now have a mechanism to find any confidence region [Wiki] of level γ. It works – when n is not too small – as follows:

- **Step 1**. Let (p_0, q_0) be the estimator of (p, q), computed on your observations set with Formula (26).
- **Step 2**. Find all (x, y)'s satisfying $H_n(x, y, p, q) = G_\gamma$, where (p, q) is replaced by (p_0, q_0). These (x, y)'s form the boundary of your confidence region in the proxy space.
- **Step 3**. Apply the inverse mapping described earlier (see Table 2 and spreadsheet section below) to map (x, y) to (λ, s). Do it for all (x, y) on the boundary obtained in step 2.

The resulting (λ, s)'s obtained in step 3 form the boundary of your confidence region in the original parameter space. The methodology described here is generic and applicable to any estimation problem involving multidimensional parameters, regardless of the complexity. In Exercise 27, I introduce a new type of confidence region called dual confidence region, obtained by swapping the roles of (p, q) and (x, y) in Formula (27). This new concept is also discussed here and here.

Again, the choice of p, q as proxy statistics is not ideal, but it leads to an easy implementation in Excel, offering educational value. A different choice may lead to more narrow confidence regions, that is, a higher confidence level γ. Or to put in another way, it may require a smaller sample size [Wiki] (that is, smaller observations sets) to produce the same level of confidence. This is true if you choose proxy statistics that, unlike p and q, are independent.

Remark: Most authors use $1 - \alpha$ for the confidence level, based on a long tradition. A different but related concept called "significance level" is denoted as α: it is technically defined as "one minus the confidence level" [Wiki]. Then the "critical value" is denoted as Z_α. Here, I use γ instead of α or $1 - \alpha$, and a single term "confidence level" to avoid confusions.

Observed Data, Simulations

I produced the observation sets using simulated Poisson-binomial processes of intensity $\lambda = 1.4$ and scaling factor $s = 0.6$. The simulations and all the computations are in the Excel spreadsheet discussed at the end of Section 3.1. The simulations are useful in different ways:

- It helps you assess if the methodology works. It does not work if the estimates are occasionally or frequently very different from the actual values used in the simulation, or if increasing the sample size does not lead to convergence to the actual values (unless you use a flawy pseudo-random number generator). My methodology passes these tests.
- It helps you decide which proxy statistics to choose from when faced with multiple options. Choose the one consistently requiring the smallest sample size, if there is such a clear winner. Simulations also have a benchmarking potential, by allowing you to compare your method with other solutions.
- It helps you assess when boundary effects become an issue, see Section 3.1.2.

In my example, boundary effects were never an issue because n was large enough, given the small value of s: the approximation errors intrinsic to the method were much larger than the minuscule bias caused by ignoring boundary effects. Still, it is always good practice to quantify all potential sources of bias. In some occasions, the pseudo-random number generator itself was one of the major sources of inaccuracies (see Section 3.6.1), until it got identified and replaced by a better one. In other occasions, roundoff errors caused by numerical instability were to blame. It got fixed by using more stable computations or high precision computing [Wiki].

Hierarchy of Estimation Methods

The estimation method previously discussed is at level 3 on a scale from 1 to 4. The levels are as follows:

- **Level 1**. An obvious, natural statistic makes sense to estimate the parameter of interest: for instance, the average computed on the observations, to estimate the theoretical mean value. In addition, an exact or asymptotic formula exists, to determine the bounds of a confidence interval of level α, given a sample size of n. This requires that a stochastic model underlined by a parametric family of distributions, is a potential fit for the data. Then the estimation procedure consists of finding the parameter value achieving the best fit, and computing the confidence interval.
- **Level 2**. The situation is identical to level 1, except that no simple formula exists for the distribution. Then one uses simulations and numerical approximations instead.
- **Level 3**. No natural statistic exists to estimate the parameter. At this level, generally (but not always) no simple exact formula exists to compute the theoretical distribution. An asymptotic formula valid as $n \to \infty$, may still be available. Typically, simulations are required. A proxy statistic can be used to estimate the quantity of interest. In my example, this is true for the parameter s: see the methodology discussed to build confidence regions. This level corresponds to model-free estimation, in the sense that no model-based formula, whether exact, approximated or asymptotic, is used to perform the computations and derive the confidence regions.
- **Level 4**. This level is known as true model-free, or data-driven inference [Wiki]. There may be no natural or simple stochastic model explaining the patterns in the the observations, and the parameter of interest can be rather obscure. In this case, one can use resampling techniques [Wiki] to produce the simulated data needed to compute confidence regions or to perform statistical tests. This is explained in my book "Statistics: New Foundations, Toolbox, and Machine Learning Recipes" [37].

With my estimation procedure, if you only use resampled observations to produce the clusters in Figure 10, you can obtain a confidence region for (p, q). However, it is impossible to retrieve (λ, s) since this parameter (the scaling factor s in particular) is associated to the model, while (p, q) is not. The confidence region of level γ would be (say) an ellipse centered at (p_0, q_0) in the proxy space, containing a proportion γ of the cluster centers. The value (p_0, q_0) would still be computed the same way, using Formula (26). Somehow though, you would still be able to test whether two observations sets have the same (λ, s). They would have the same (λ, s) – something that you can not test without the model – if and only if they have the same (p, q) – something that you can easily test without using any model.

Confidence regions and statistical tests based on resampled observations may be slightly biased or asymptotically unbiased. Sometimes the bias can be quantified and corrected. Popular methods include bootstrapping [Wiki] and k-fold cross-validation [Wiki]. Also see "A New Distribution-Free Approach to Constructing the Confidence Region for Multiple Parameters" [43], available online here.

Spreadsheet

The spreadsheet is available on my GitHub repository, here: `PB_independence.xlsx` (click on the link to access it). Look for the `Confidence_Region` tab. I simulated $N = 10{,}000$ observations sets, each with n observations. I used the values p_0, q_0 in cells `B1`, `B2`, and a bivariate Bernoulli model with these values as parameters, to generate the observations. The source code related to the Bernoulli model is in column `Y`. The Bernoulli model is described in Exercise 27, as well as here and here.

Each row in the spreadsheet table represents one of the N sets, with the estimated proportions p, q in columns `D` and `E`, then $\sigma_p, \sigma_q, \rho_{p,q}$ in columns `H`, `I`, `J`, and G_γ in column `G`. These quantities were computed using the source code in column `Y`, based on Formulas (26) and (27). The rows are sorted by the values in column `G`. The confidence region featured in Figure 11 corresponds to the (p, q)'s in the first 9000 rows, after the sorting in question. Thus the confidence level is a $\gamma = 90\%$. The corresponding $G_\gamma = 4.595$ is in cell `G:9001`.

3.1.2 Periodicity and Amplitude of Point Counts

Let (X_k), with $k \in \mathbb{Z}$, represents the points of a one-dimensional Poisson-binomial process of intensity λ and scaling factor s. We are interested in point counts $N_\tau(t) = N[B_\tau(t)]$ in the interval $B_\tau(t) = [t, t + \tau[$. Let

$$\phi_\tau(t) = \mathrm{E}[N_\tau(t)].$$

By virtue of Theorem 4.1, $\phi_\tau(t) = 1$ if $\tau = 1/\lambda$. More generally, regardless of τ, the function $\phi_\tau(t)$ is periodic of period $1/\lambda$. That is, $\phi_\tau(t) = \phi_\tau(t + 1/\lambda)$. This latter statement is also true for $\text{Var}[N_\tau(t)]$, $P[N_\tau(t) = 0]$, and $P[N_\tau(t) = 1]$. This fact is trivial if you look at Formulas (4), (5), (6) and (7), used to compute the four quantities in question.

The amplitude of the oscillations is extremely small even with a scaling factor as low as $s = 0.3$ (assuming F is logistic). It quickly tends to zero as $s \to \infty$. So, the process is almost stationary unless s is very close to zero. Thus, in most inference problems, the choice of the (non-overlapping) intervals has very little impact. In particular, $\phi_\tau(t) \approx \lambda\tau$. The small amplitude of $\phi_\tau(t)$ is pictured in Figure 12.

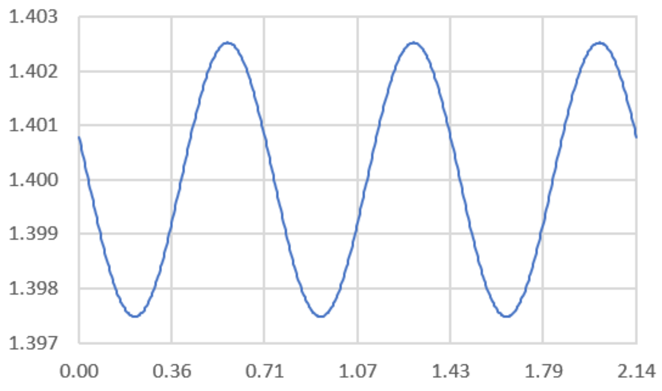

Figure 12: Period and amplitude of $\phi_\tau(t)$; here $\tau = 1, \lambda = 1.4, s = 0.3$

Assuming that $\phi_\tau(t)$ is constant and equal to $\lambda\tau$ results in a tiny error, unless s is very close to zero. To the contrary, boundary effects are a bigger source of bias, this time when s is large. Simulations can quantify the amount of bias, see Section 3.5. See also the spreadsheet section below.

Spreadsheet

The functions $\phi_\tau(t) = \text{E}[N_\tau(t)]$, $\text{Var}[N_\tau(t)]$, $P[N_\tau(t) = 0]$ and $P[N_\tau(t) = 1]$ are tabulated in the spreadsheet `PB_independence.xlsx`. See columns `D` to `I` in the `Periodicity` tab. The parameters are $\lambda = 1.4$ (cell `B1`) and $s = 0.3$ (cell `B2`). The source code to produce this table is in column `AI`. Here $\tau = 1$.

Also, columns `U` to `Z` contains the computations to estimate λ based on a realization of a Poisson-binomial process in column `S`. I generated $2n + 1$ points, with $n = 5000$. The estimator, denoted as λ_0, is in column `Z`. I computed different versions of $\lambda_0 = N_\tau(t)/\tau$, based on different values of t and τ. The point counts $N_\tau(t)$ are computed on the simulated realization. The true value $\lambda = 1.4$ (used for the simulation in column `S`) is stored in cell `B4`, while $s = 12$ is stored in cell `B5`. The purpose is to find optimum t, τ that minimize the boundary effects, to get an unbiased estimator λ_0 of the intensity λ.

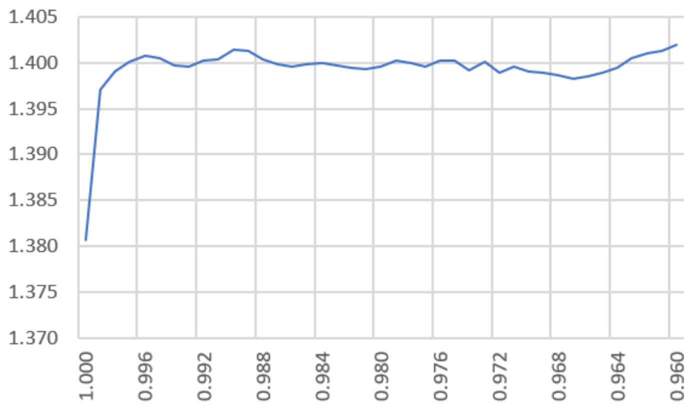

Figure 13: Bias reduction technique to minimize boundary effects

Figure 13 shows how λ_0 (on the Y axis) varies depending on the choice of a parameter α (on the X axis, and also in column `U` in the spreadsheet). The parameter α, with $0.96 < \alpha \leq 1$ in the picture, determines the interpercentile range $[L_\alpha, U_\alpha] = [t, t+\tau[$ used to compute λ_0. When $\alpha = 1$ (the leftmost position on the X axis), L_α is the minimum, and U_α is the maximum value among the points of the process. The bias is also maximum. The smaller α, the fewer points used to compute λ_0, and the further away we are from the boundaries, thus reducing the bias to almost zero if α is small enough. Yet the smaller α, the more unstable λ_0 is. Thus one needs to find the right balance between a too large and a too small value of α.

In my example, if you look at Figure 13, $\alpha = 0.992$ achieves this goal, yielding an estimate $\lambda_0 = 1.400$ correct to three digits. Note that $\alpha = 1$ yields a biased value of λ_0 between 1.380 and 1.390 depending on the simulation (close to 1.380 in Figure 13). A technique such as the automated elbow rule, described in Section 3.4.4, can be used to detect the optimum α, and thus the optimum λ_0.

3.1.3 A New Test of Independence

You use this kind of tests for instance to assess whether the point counts $N(B)$ in various non-overlapping domains B are independent or not. Generally, one works with domains of same area. The most popular test of independence is the χ^2 (chi-squared) test [Wiki]. One drawback of χ^2 is that it requires binning the data. The bin size can not be too small, and the bins may be arbitrary. My approach is different, and avoids this problem. It is also well suited to detect small deviations from independence.

It works as follows. I compare the empirical distribution of count frequencies with what it should be if the counts were independent. I offer two solutions: one based on the R-squared [Wiki], and one based on the Kolmogorov-Smirnov statistic [Wiki]. The latter is similar to the approach discussed by Zhang in his article "A Kolmogorov-Smirnov type test for independence between marks and points of marked point processes" [82], available online here.

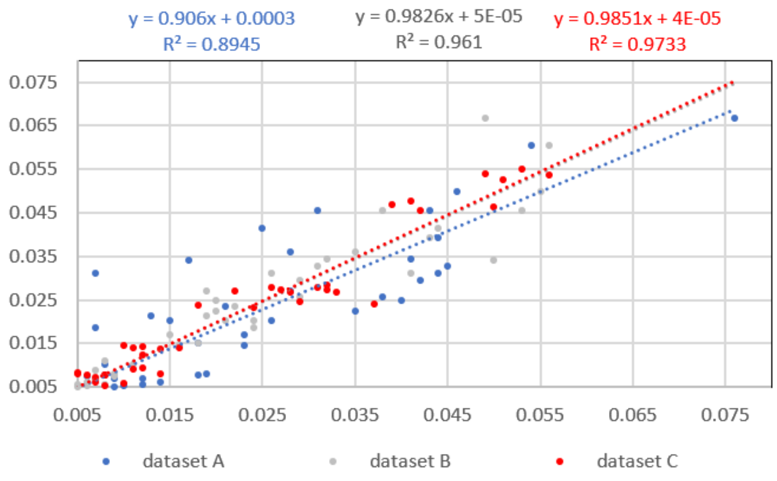

Figure 14: A new test of independence (R-squared version)

Let (X_k) be the points of a Poisson-binomial process M_A of intensity $\lambda = 1$ and scale factor $s = 0.7$, with a logistic F. Exercise 10 shows – using theoretical arguments – that the point counts are not independent. Here I establish the same conclusion via statistical testing. The purpose is to illustrate how the test works, so that you can use it in other contexts. I chose three intervals $B_1 = [-1.5, -0.5[$, $B_2 = [-0.5, 0.5[$, and $B_3 = [0.5, 1.5[$. The data consists of $m = 1000$ realizations of the process in question, each one consisting of 41 points X_k, $k = -20, \dots, 20$. The number 41 is large enough in this case, to eliminate boundary effects. The data, computations and results are in the spreadsheet `PB_independence.xlsx`, described later in this section.

The point counts attached to a realization ω of the point process, is denoted as N_ω. The aggregated point count over the m realizations is denoted as N, and the set of m realizations is denoted as Ω. Now, for $i = 1, 2, 3$ and $j_1, j_2, j_3 \in \mathbb{N}$, I can define the following quantities:

$$p_i(j) = \frac{1}{m} \sum_{\omega \in \Omega} \chi[N_\omega(B_i) = j],$$

$$p(j_1, j_2, j_3) = \frac{1}{m} \sum_{\omega \in \Omega} \prod_{i=1}^{3} \chi[N_\omega(B_i) = j_i], \tag{29}$$

$$p'(j_1, j_2, j_3) = \frac{1}{m^3} \prod_{i=1}^{3} \sum_{\omega \in \Omega} \chi[N_\omega(B_i) = j_i], \tag{30}$$

where χ is the indicator function [Wiki]. For instance, $p_1(3) = 0.043$ means that in 43 realizations out of $m = 1000$, the domain B_1 contained exactly 3 points. Also, $p'(j_1, j_2, j_3) = p_1(j_1)p_2(j_2)p_3(j_3)$. The three point counts $N(B_1), N(B_2), N(B_3)$ are independently distributed if and only if Formulas (29) and (30) represent the same quantity when $m = \infty$. In other words, the three point counts are independently distributed if $p \to p'$ pointwise [Wiki], as $m \to \infty$.

To avoid future confusion, p and p' are denoted as p_A and p'_A to emphasize the fact that they are attached to the process M_A. To test for independence, I simulated m realizations of a sister point process M_B: one with the same marginal distributions for the three point counts, using the estimates $p_i(j)$ obtained from M_A, but this time with guaranteed independence of the point counts, by design. Likewise, I define the functions p_B and p'_B. Let ρ_A be the correlation between p_A and p'_A, computed across all triplets satisfying

$$\min\{p_A(j_1, j_2, j_3), p'_A(j_1, j_2, j_3)\} > \epsilon.$$

I chose $\epsilon = 0$. In my example, there were fewer than $7 \times 7 \times 7 = 343$ such triplets. Finally, the statistic of the test is ρ_A^2.

Results and Interpretation

In the spreadsheet PB_independence.xlsx, the tab Dataset_A corresponds to M_A, and Dataset_B corresponds to M_B. The same computations are done in tab Dataset_C for another point process M_C, identical to M_A except that this time $s = 4$. With such a "large" s, M_C is not that different from a stationary Poisson point process: in particular, the point counts are almost independent (no statistical test could detect that they are not, unless using extremely large samples).

The main findings are displayed in Figure 14. Blue represents the M_A process, gray represents M_B, and red represents M_C. Each blue dot corresponds to a vector (p_A, p'_A) attached to a particular (j_1, j_2, j_3). In case of perfect independence, all the dots should be on the main diagonal. Blue dots are two far away from the main diagonal, and thus the point counts in M_A are not independent. To the contrary, M_B (supposed to exhibit independence by construction) and M_C (known from theory to exhibit near-independence) are close enough to each other and to the main diagonal. If you repeat the experiment with M_B a hundred times, you will get a hundred gray regression lines, generating a confidence curve for the test. Note that the R^2 displayed for the three regression lines in Figure 14, are identical to $\rho_A^2, \rho_B^2, \rho_C^2$, confirming the somewhat poor performance of M_A. The slope of the regression line is also an indicator of lack of independence, if it is not close enough to one. Again, M_B is the loser here, when measured against the slope. The intercept of the regression line (when different enough from zero) further confirms this.

A version of this test, available in the spreadsheet, relies on the Kolmogorov-Smirnov statistics instead of the R-squared. It works with aggregated rather than raw frequencies. In short, you replace the empirical probabilities p_A, p'_A (the frequencies) by empirical aggregated probabilities P_A, P'_A, that is, the empirical distributions. The statistic of the test is the uniform norm [Wiki] $\delta_A = ||P_A - P'_A||_\infty$. It leads to the same conclusion. Since the argument of the functions p_A, p'_A are the triplets (j_1, j_2, j_3) and are unordered, there are many different ways to build the empirical distribution. However, the differences among these constructions are minuscule. See also the section "Interactions in Point Pattern Analysis" in [40], available online here.

About the Spreadsheet

The interactive spreadsheet is on my GitHub repository: see PB_independence.xlsx. The Summary tab controls the parameters s, λ, and the upper/lower bounds of the intervals B_1, B_2, B_3. It also contains the results: the R-squared's $\rho_A^2, \rho_B^2, \rho_C^2$ respectively in cells B11, C11, D11, and the Kolmogorov-Smirnov statistics $\delta_A, \delta_B, \delta_C$ respectively in cells B12, C12, D12. Columns J, K, L represent the triplets (j_1, j_2, j_3), also available in concatenated format in column I. For the M_A process, the empirical probabilities p_A, p'_A are in columns Q, R, and the empirical distributions P_A, P'_A are in columns S, T. For M_B and M_C, the corresponding values are in columns Z to AD and AI to AM respectively.

In the Dataset_A and Dataset_C tabs, each row (except the first one) represents a realization of the underlying point process, respectively M_A and M_C. The 41 points of each realization are in columns F to AT. The first row (same columns) stores the indices of the points in question, in the index space. I used the logistic distribution for F.

The Dataset_B tab corresponds to M_B. It is organized differently. The actual points of each realization are not computed as they are not needed this time. Thus they are not in the spreadsheet. Instead, point counts summarizing each "unobserved" realization are in columns I, J, K, corresponding respectively to B_1, B_2, B_3. Columns Q and R, representing the values of p_B and p'_B (with the argument in column F), are derived from these counts. Remember that M_B was designed so that (1) $p'_B = p'_A$ and (2) the point counts $N(B_1), N(B_2), N(B_3)$ are independent.

3.2 Estimation of Core Parameters

It is assumed that the point process covers the entire state space $\mathbb{R}$ or $\mathbb{R}^2$ with infinitely many points, and that only a finite number of points are observed through a finite (typically rectangular) window or interval. Here I focus on the one-dimensional case. For processes in two dimensions, see Section 3.4.2.

3.2.1 Intensity and Scaling Factor

In one dimension, the two most fundamental parameters are the intensity λ and the scaling factor s. The standard estimator of λ proposed here is asymptotically unbiased [Wiki], see Section 3.1.2. For a more generic, model-free method yielding an unbiased estimator simultaneously for s and λ, along with confidence regions, see Section 3.1.1. The goal of this section is to offer efficient estimators, easy to compute, and taking advantage of the properties of the underlying model.

Estimation of λ

There are various ways to estimate the intensity λ (more specifically, λ^d in d dimension) using interarrival times T, nearest neighbors (in two dimensions) or the point count $N(B)$ computed on some interval B. A good estimator with small variance, assuming boundary effects are mitigated (see Section 3.5), is the total number of observed points divided by the area (or length, in one dimension) of the window of observations.

Another estimator is based on Theorem 4.3: the expected value of the interarrival time is $1/\lambda$. Thus, if you average all the interarrival times accross all the observed points (called events in one dimension), you get an unbiased estimator of $1/\lambda$. Its multiplicative inverse will be a slightly biased estimator of λ; if the number of points is large enought (say > 50), the bias is negligible.

Estimation of s

Once λ has been estimated, the scaling factor s can be estimated by leveraging Theorem 4.2. The strategy is as follows. Let λ_0 be your estimate of λ. By virtue of Theorem 4.2, the interarrival times satisfy $\text{E}[T^r(\lambda, s)] = \text{E}[T^r(1, \lambda s)]/\lambda^r$ for any $r > 0$. This result does not depend on the distribution F.

With $r = 2$, let

- τ_0 be your estimate of the squared interarrival times (the average squared value), computed on your data set,
- $\tau' = (\lambda_0)^r \cdot \tau_0$, where λ_0 is your estimate of λ (see above subsection),
- s' be the solution to $\text{E}[T^r(1, s')] = \tau'$.

Then $s_0 = s'/\lambda_0$ is an estimate of s.

Example: Here F is the logistic distribution, and I chose $r = 2$. Any $r > 0$ except $r = 1$ would work. If $\lambda_0 = 1.45$ and $\tau_0 = 0.77$, then $\tau' = (\lambda_0)^2\tau_0 = 1.61$. Looking at the $\text{E}[T^2(1, s')]$ table, to satisfy $\text{E}[T^2(1, s')] \approx 1.61$, you need $s' = 0.65$. Thus $s_0 = s'/\lambda_0 = 0.45$. These numbers match those obtained by simulation. To view or download the table, look at the $\text{E}[T^2]$ tab in `PB_inference.xlsx`.

The equation $\text{E}[T^2(1, s')] = \tau'$, where s' is the unknown, can be solved using numerical methods. The easiest way is to build a granular table of $\text{E}[T^2(1, s)]$ for various values of s, by simulating Poisson-binomial processes of intensity $\lambda = 1$ and scaling factor s. Then finding s' consists in browsing and interpolating the table in question the old fashioned way, to identify the value of s closest to satisfying $\text{E}[T^2(1, s)] = \tau'$. This can of course be automated. There are two ways to perform the simulations in question:

- generating one realization of each process with a large number of points (that is, one realization for each $0 < s < 20$ with $\lambda = 1$ and s increments equal to 0.01),
- or generating many realizations of each process, each one with a rather small number of points.

Either way, the results should be almost identical due to ergodicity if the same F is used in both cases. The simulations also allow you to compute the theoretical variance of the estimators in question (at least a very good approximation). This is useful when multiple estimators (based on different statistics) are available, to choose the best one: the one with minimum variance. Simulations also allow you to compute confidence intervals for your estimators, as discussed in Section 3.1. The source code for the simulations can be found in Section 6.2.

Alternative Estimation Method for s

It is also possible to use the point count $N(B)$ to estimate s. The idea is to partition the state space (the real line in one dimension, where the points reside) into short intervals $B_k = \left[\frac{k}{\lambda}, \frac{k+1}{\lambda}\right[$, $k = 0, \pm 1, \pm 2$ and so on, covering the observed points; beware of the boundary effect. This assumes that λ is known or estimated. Let $N_k = N(B_k)$ be the random variable counting the number of observed points in B_k. We have $\text{E}[N_k] = 1$. Also

$\text{Var}[N_k] \leq 1$ does not depend on k thanks to the choice of B_k (see Section 3.1.2). The variance is maximum and equal to one when $s = \infty$.

It is possible, for any value of s and λ, to compute the theoretical variance $v(\lambda, s) = \text{Var}[N_k]$ using either simulations or Formula (5) with $a = 0$ and $b = 1/\lambda$. It slightly depends on F, but barely. Now compute the empirical variance of N_k as the average $(N_k - 1)^2$ across all the B_k's, based on your observations, assuming λ is known or estimated. This empirical variance is denoted as $v_0(\lambda)$. The estimated value of s is the the one that makes the empirical and theoretical variances identical, that is, the *unique* value of s that solves the equation $v(\lambda, s) = v_0(\lambda)$. This method easily generalizes to higher dimensions, see Section 3.4.2. The fact that $\text{E}[N_k] = 1$ is a direct consequence of Theorem 4.1.

See the N_k tab in `PB_inference.xlsx`, for a Poisson-binomial process simulation with a generalized logistic F, and computation of $\text{E}[N_k]$ and $\text{Var}[N_k]$ in Excel. You can download the spreadsheet from the same location.

3.2.2 Model Selection to Identify F

It is more difficult if not impossible to retrieve the distribution F attached to each point X_k. However, see Exercise 6. In many cases, two different F's result in essentially the same model, causing model identifiability issues. The situation if much easier if s is very small, small enough that $|X_k - \frac{k}{\lambda}| < \frac{1}{2\lambda}$ for most $k \in \mathbb{Z}$. Then the index attached to a point X, usually unknown, is now equal to

$$L(X) = \arg\min_{k \in \mathbb{Z}} \Big| X - \frac{k}{\lambda} \Big|.$$

That is, $X = X_k$ with $k = L(X)$. See definition of $\arg\min$ here. This assumes that λ is known or estimated. In this particular situation, assuming s is also known or estimated, the empirical distribution of $s \cdot (X - L(X))$ computed over many points X, converges to F as the number of observed points tends to infinity. See also Section 4.7 about the hidden process, and Exercise 12.

A more practical situation is when one has to decide which F provides the best fit to the data, given a few potential candidates for F. In that case, one may compute (using simulations) the theoretical expectation $\eta(r, \lambda, s, F) = \text{E}[T^r(\lambda, s)]$ as a function of $r > 0$ for various F's, and find which F provides the best fit to the estimated $\text{E}[T^r(\lambda, s)]$, denoted as $\eta_0(r, \lambda, s, F)$ and computed on the data (the expectation being replaced by an average when computed on the data). By best fit, I mean finding F that minimizes (say)

$$\gamma(F) = \int_0^2 |\eta(r, \lambda, s, F) - \eta_0(r, \lambda, s, F)| dr. \tag{31}$$

Again, s and λ should be estimated first. However, a simultaneous estimation of λ, s, F is feasible and consists of finding the parameters λ, s, F minimizing $\gamma(F)$, now denoted as $\gamma(\lambda, s, F)$. See Section 3.2.1 to estimate λ and s separately: this stepwise procedure is simpler and less prone to overfitting [Wiki].

The estimation technique introduced here, especially Formula (31), is sometimes referred to as minimum contrast estimation. See slides 114–116 in the presentation entitled "Introduction to Spatial Point Processes and Simulation-Based Inference", by Jesper Møller [58], available online here or here.

3.2.3 Theoretical Values Obtained by Simulations

This section highlights some simulation results obtained with the source code in Section 6.2 to compute moments $\text{E}[T^r]$ of the interarrival times $T = T(\lambda, s)$ for various λ, s as well as statistics related to the point count random variable $N(B)$, where $B = [a, b]$ is an interval. More such results are displayed in Figure 1, where a Cauchy, uniform, and logistic F are compared. The goal is to:

- show that except if F has a finite support or s is very small, the choice of F has very little impact (see Figure 1),
- show how fast the Poisson-binomial process converges to a stationary Poisson process as s increases (see Figure 1),
- show that any point of the process can be used to compute the *theoretical distribution* of T, thus choosing X_0 or any X_k, or averaging over many points, yields the same theoretical distribution (see Table 4),
- show that you can use one realization of the process with many points, or many realizations of the process, each with few points, to compute the theoretical distribution of T.

The last fact in the above list illustrates the ergodicity of T.

	Formula	Value	Uniform	Logistic	Cauchy
	$s=\infty$	$s=\infty$	$s=39.85$	$s=39.85$	$s=39.85$
$\text{E}[N(B)]$	$\lambda\mu(B)$	$3/2$	1.5019	1.5000	1.4962
$\text{Var}[N(B)]$	$\lambda\mu(B)$	$3/2$	1.4738	1.4906	1.4872
$\text{P}[N(B)=0]$	$e^{-\lambda\mu(B)}$	0.2231	0.2196	0.2221	0.2230
$\text{E}[T]$	$1/\lambda$	1	1.0003	0.9999	1.0010
$\text{Var}[T]$	$1/\lambda^2$	1	0.9680	0.9888	1.0029
$\text{E}[\sqrt{T}]$	$\frac{1}{2}\sqrt{\pi/\lambda}$	0.8862	0.8865	0.8862	0.8873

Table 3: Poisson process ($s=\infty$) versus F_s (with $s=39.85$)

Table 3 summarizes some statistics produced with the source code in Section 6.2, with $\lambda=1, r=1/2$ and $B=[a,b]$. Here, $a=-0.75$ and $b=0.75$. The notation $\mu(B)$ stands for $b-a$. In two dimensions, it represents the area of the set B (typically, a square or a circle). In one dimension, when $s=\infty$, $N(B)$ has a Poisson distribution of expectation $\lambda\mu(B)$, and T has an exponential distribution of expectation $1/\lambda$. The limiting process is a stationary Poisson process of intensity λ. The exact formula for $\text{E}[\sqrt{T}]$, when $s=\infty$, was obtained with the online version of Mathematica: you can check the computation, here. In general, convergence to the Poisson process, when $s\to\infty$, is slower and more bumpy if F is uniform, compared to using a logistic or Cauchy distribution for F.

k (in X_k)	-5	-4	-3	-2	-1	0	1	2	3	4	5
$E[T_k]$	0.99	0.98	1.01	0.99	1.01	1.00	1.00	1.02	1.00	1.00	1.01
$E[T_k^{1/2}]$	0.90	0.90	0.90	0.91	0.90	0.91	0.91	0.91	0.90	0.90	0.91
$E[T_k^{3/2}]$	1.24	1.24	1.24	1.27	1.22	1.29	1.27	1.26	1.26	1.24	1.27
$E[T_k^2]$	1.70	1.68	1.70	1.75	1.67	1.79	1.76	1.70	1.74	1.71	1.75

Table 4: Moments $\text{E}[T_k^r]$ of interarrival times, for $r=0.5,\dots,2$ and $k=-5,\dots,5$

Table 4 displays various moments obtained by simulation, from averaging T_k^r across 10^4 realizations of a Poisson-binomial process with a logistic F and $s=\lambda=1$, for small values of k, yielding about 2 digits of accuracy. Each realization consisted of $2n+1$ points $X_{-n}, X_{-n+1},\dots,X_0,\dots,X_{n-1},X_n$, with $n=30$ large enough to avoid significant boundary effects (see Section 3.5). The interarrival time T_k was defined as the distance between X_k, and its closest neighbor X'_k to the right. The purpose was to check whether the choice of k matters. The conclusion from looking at the table, is that it does not matter. This empirically justifies the choice $k=0$ in our definition of T in Section 1.2.

Another way to measure T is by averaging the various $T_k=X'_k-X_k$, say for $-10^4<k<10^4$, measured on a single realization of the same Poisson-binomial process, with a very large n, say $n=3\times10^4$. Here X'_k is the closest neighbor to X_k, to the right on the real axis. It yields the same result. The theoretical value for $r=1$ is $\text{E}[T]=1/\lambda$, according to Theorem 4.3. Also for $r=2$, the theoretical value if $s=\infty$ is $\text{E}[T^2]=2/\lambda^2$ due to the Poisson process approximation. The value reported in Table 4 is around 1.72, and this is for $s=1$. We are not that far from the Poisson limit!

3.3 Hard-to-Detect Patterns and Model Identifiability

Poisson-binomial and related point processes such as m-interlacings, exhibit many hard-to-detect patterns. Some can not even be detected with statistical tests. Depending on model parameters, many are not visible to the naked eye. In some cases, this is due to model identifiability: two apparently different models, with different sets of parameters, are statistically identical and indistinguishable from each other. Most of the times though, the differences are real but subtle or imperceptible. To the contrary, on occasions, the naked eye perceives differences when there are none, akin to visual illusions.

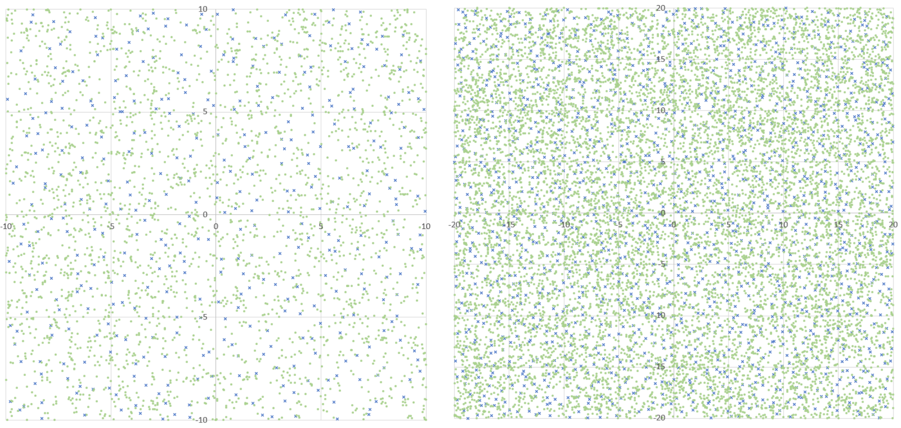

Figure 15: Radial cluster process ($s = 0.5, \lambda = 1$) with centers in blue; zoom in on the left

In this section, I explore some these peculiarities. As a starter, let's look at Figures 3 and 4. They clearly represent two distinct models: lattice structure, versus random point distribution. But what about Figures 15 versus 16? Actually, all four feature the same model. The only difference is the choice of the scaling factor s. The first two represent two extremes: $s = 0.2$ versus $s = 2$. But the last two correspond to in-between cases ($s = 0.5$ versus $s = 1$), and look similar. Also, unless you have experience dealing with these processes, it is not easy to tell whether or not the point pattern in Figure 16, despite looking a bit more "random" than in Figure 15, corresponds to pure randomness (a stationary Poisson process). The answer is negative despite the appearances: the points are too evenly spread to represent pure randomness.

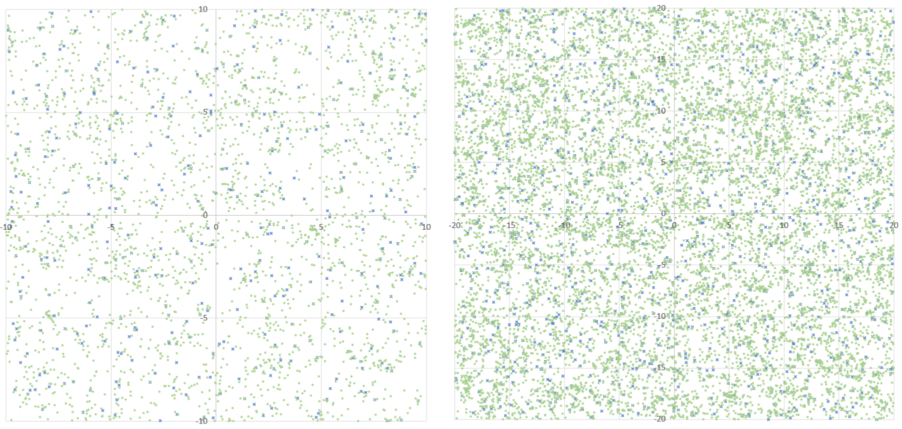

Figure 16: Radial cluster process ($s = 1, \lambda = 1$) with centers in blue; zoom in on the left

Other examples of hard-to-detect differences include:

- Discriminating between two different F's (the distribution attached to the points), for instance logistic versus Gaussian or Cauchy, unless s is very small.
- If s is large, the process is hard to distinguish from a stationary Poisson process: see Figure 4.
- Point count statistics (expectation, variance and so on) are periodic, but amplitudes are extremely small.
- The cluster structure in m-interlacings may be invisible unless some transformation is applied: see left plot in Figure 17. Nearest neighbor distances are generally better at detecting differences, compared to point counts.
- Unless s is very small, it may be impossible to detect if the underlying lattice space is square or hexagonal, or if we are dealing with an m-interlacing or a m-mixture.

To the contrary, in some cases, the naked eye perceives non-existent differences. For instance, the fact that the right plot in Figure 2 has fewer points than the left plot, when in fact they both have the same number. In fact, the Poisson-binomial model is a good framework to test and benchmark statistical techniques in contexts that require a very high level of precision. For instance, those aimed at detecting exoplanets, early signs of cancer, or subtle patterns in the stock market.

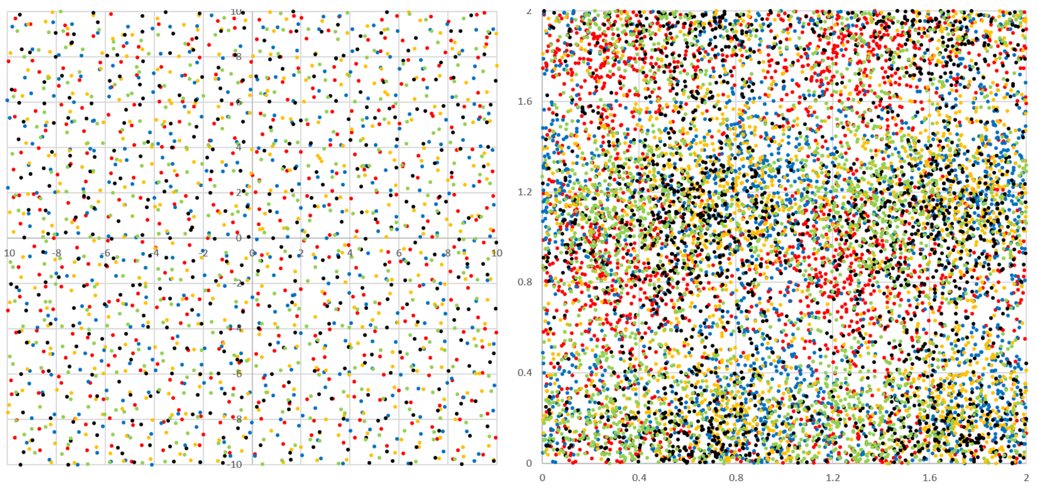

Figure 17: Realization of a 5-interlacing with $s = 0.15$ and $\lambda = 1$: original (left), modulo $2/\lambda$ (right)

3.4 Spatial Statistics, Nearest Neighbors, Clustering

I already discussed spatial processes based on the Poisson-binomial point process model, using radial distributions and an infinite number of clusters, in Section 2.1. The cluster processes investigated in this section are different (see Exercise 14): it is a superimposition of m shifted Poisson-binomial processes called an m-interlacing, or a mixture of m such processes, called an m-mixture. It represents a structure with m clusters. They were introduced in Section 1.5.3 and 1.5.4, and further investigated in Exercises 18 and 19. Simulation of m-interlacings is straightforward, using Formulas (8) and (9). The concept is also very intuitive. A realization with $m = 5$ is shown in Figure 17, with a different color assigned to each individual process of the m-interlacing. Full source code is available in Part 2 of Section 6.4.

The main purpose is to discuss a new type of GPU-based clustering algorithm (Section 3.4.3), using image filtering techniques taking place in the graphics processing unit (GPU) [Wiki] to accelerate the computations [Wiki]. In addition, we are interested in estimating the parameters of the model (Section 3.4.2), including automated detection of the number of clusters (Section 3.4.4) using a modern black-box version of the elbow rule [Wiki].

There are three kinds of clustering: Supervised clustering [Wiki], unsupervised clustering [Wiki], and semi-supervised clustering. Shervine Amidi's cheatsheets related to his machine learning class CS 229 at Stanford university, provide easy-to-read, very useful summarized information about the various techniques. You can access them on his webpage or on Github. Clustering is one of the main techniques in machine learning [Wiki]. It is a good candidate for machine learning automation (abbreviated as AutoML), a field of AI, especially using the methodology described in this section.

3.4.1 Stochastic Residues

Each individual process of the combined point process (the m-mixture or m-interlacing) has its own shift vector, which determines the center of a cluster. By translation, the cluster is replicated around each lattice location in the lattice space, and thus in the state space as well. As a result, for statistical inference, it is customary to study the process (the observed data) modulo $2/\lambda$ or $1/\lambda$, where statistical patterns are magnified and easier to detect. By modulo $2/\lambda$, I mean the following: instead of studying the original points (X, Y), we focus on $(X \bmod 2/\lambda, Y \bmod 2/\lambda)$. The transformed data, after the modulo operation, is called the residual data, or stochastic residues. After the modulo operation, we are left with m clusters: one per individual process. The fact that there are $m = 5$ clusters (albeit with huge overlap) in Figure 17 is apparent on the right plot featuring the residues, but not on the left plot. In Section 3.4.3, we shall see how to identify these clusters. Typically, in the context of unsupervised clustering, we don't known which individual process a point of the combined process belongs to.

Remark: The modulo operator is defined as $\alpha \bmod \beta = \alpha - \beta \cdot \lfloor \alpha/\beta \rfloor$, where the brackets represent the floor function (also called integer function [Wiki]). It is identical to the one used in modular arithmetic [Wiki], except that here, α, β are usually real numbers rather than integers.

3.4.2 Inference for Two-dimensional Processes

Let us assume for now that we are dealing with a single two-dimensional Poisson-binomial point process. Some of the methodology discussed in Section 3.2 for the one-dimensional case can be generalized to higher dimensions.

The point count in a square of side $1/\lambda$ has expectation equal to one, according to a multidimensional version of Theorem 4.1. So, one way to estimate λ is to partition the window of observations W into small squares $B_{h,k}(\lambda) = \Big[\frac{h}{\lambda}, \frac{h+1}{\lambda}\Big[\times \Big[\frac{k}{\lambda}, \frac{k+1}{\lambda}\Big[$ for various values of the (unknown) λ, compute the number of points $N_{h,k}(\lambda)$ (called point count) in each of these squares, and find λ that minimizes the empirical variance

$$v(\lambda) = \sum_{h,k} \Big(N_{h,k}(\lambda) - 1\Big)^2$$

computed on the observations. The sum is over $h, k \in \mathbb{Z} \cap W'$, where W is the window of observation, and W' is slightly smaller than W to mitigate boundary effects. In short, your estimate of the intensity λ is defined as $\lambda_0 = \arg\min_{\lambda} v(\lambda)$.

The benefit of this approach is that it also allows you to easily estimate the scaling factor s. Since $v(\lambda)$ also depends on the unknown s, let's denote it as $v(\lambda, s)$. Also, let $V(\lambda, s)$ be the theoretical variance of the point count $N(B)$ in $B = \Big[0, \frac{1}{\lambda}\Big[\times \Big[0, \frac{1}{\lambda}\Big[$, computed using simulations or via the Formula (5). The estimated value of s, assuming λ_0 is the estimate of λ, is the solution to the equation $v(\lambda_0, s) = V(\lambda_0, s)$.

Another simple estimator, this time for λ^d, is the total number of observed points in the observation window W, divided by the area of W. Here $d = 2$ is the dimension of the state space. Estimators of λ and s may also be obtained using nearest neighbor distances, just like I did with interarrival times in one dimension in Section 3.2.1. I haven't checked if the random variable S, defined as the size of the connected components associated to the undirected nearest neighbor graph (see Exercise 20), is of any use to estimate s. Confidence intervals can be built as in Section 3.1.

Other Possible Tests

Besides estimating the core parameters, many other properties or features can be tested. They are too numerous to be treated in details here, so I only provide a quick summary. See Exercise 9 for more details.

- Anisotropy: it means that the point distribution is statistically identical in any direction; an example is a cluster process with a radial point distribution around each cluster center, see section 2.1. Testing for anisotropy can be done using $\rho(z, r) = N[B(z, r)]/(\pi r^2)$ where $B(z, r)$ is a circle of radius r centered at z, and N is the point count. In case of anisotropy, and assuming r is not too small so that each circle has at least 20 points, there should be only little variations among the $\rho(z, r)$'s computed at different (z, r). Simulate a truly anisotropic process (stationary Poisson) with the same number of points in the window of observations, to find exactly what "only little variations" means.
- Stationarity: This consists of testing whether $N[B(z, r + t)] - N[B(z, r)]$ depends only on t, and not on r. Here, using squares centered at z and of side r for $B(z, r)$, would show lack of stationarity if s is small and you try different values of t, say $t = 1/(2\lambda)$ and $t = 1/\lambda$.
- Correlation: Are the X and Y coordinates correlated? The standard Poisson-binomial process assumes independence between X and Y, see Formula (2). So a non-zero correlation indicates that the data does not fit to a standard Poisson-binomial model. Using different stretching factors for the X and Y coordinates (see Section 1.5.2) can have an impact on the correlation when s is small.
- Independence: This test, discussed in Section 3.1.3, is used for instance to assess whether the point counts $N(B)$ in various non-overlapping domains B are independent or not. Generally, one uses domains of same area $\mu(B)$ for the test. In one dimension, it is also used to test whether increments (that is, successive interarrival times) are independent.
- Ergodicity: For some statistics based on simulations (as opposed to a real-life dataset), one can use a single realization of the process with many points or a large window of observations, to make inference. Or one can use many realizations, each one with few points or small window, to compute the same statistic and average the observed values across all the realizations. If the results are statistically the same in both cases, the statistic in question is ergodic, for the point process model in question. A good example is the nearest neighbor distance, between two neighbor points of the process.
- Repulsion (or attraction): An attractive point process is one where points tend to cluster together, leaving large areas empty, and some areas filled with many nearby points. An example is a cluster process. The opposite is a repulsive process: points tend to stay as far away as possible from each other. The most extreme case is when the scaling factor s is zero, as in the left plot in Figure 2. Typically, the degree of attraction is determined by s. However, a cluster process can be both: for instance, if the unobserved cluster centers come from a parent point process with a very small s.
- Number of clusters: Determining the number m of clusters in an m-interlacing (superimposition of m point processes), or the number of components in an m-mixture (mixture of m point processes), is not easy and

usually not feasible if cluster overlap is substantial, at least not exactly. This is discussed in Section 3.4.3. A black-box version of the elbow rule (the traditional tool to estimate the number of clusters) is discussed in Section 3.4.4.

- Shift vectors: They are discussed in Section 1.5.2 and 1.5.3 in the context of m-interlacings (a superimposition of m processes). Each of the m individual processes has a shift vector attached to it: it determines the position of a cluster center modulo $1/\lambda$. If these vectors are well separated and s is small, they can be retrieved. See discussion in Section 3.4.3, and Figure 19, featuring 5 different shift vectors ($m = 5$) and thus 5 clusters.
- Homogeneity and stretching: In Section 1.5.3, I mention the fact that stretched processes are not homogeneous because different intensities apply to the X and Y coordinates: observations are stretched using different stretching factors for each coordinate. More generally, the process is non-homogeneous if the intensity depends on the location in the state space. Whether the process is homogeneous or not is thus easy to test, using the point count statistic $N(B)$ computed at various locations.
- m-mixture versus m-interlacing: To decide whether you are dealing with a mixture rather than a superimposition of m point processes, one has to look at the point count distribution on a square B_λ of area $1/\lambda^2$. If there is no stretching involved, the theoretical expectation of the point count is $E[N(B_\lambda)] = m$ if the process is an m-interlacing; in that case, the number of points in each B_λ is also very stable. The first thing to do is to estimate λ (see the beginning of Section 3.4.2), then look at the empirical variance of $N(B_\lambda)$ computed on the observations. When s is small enough, $N(B_\lambda)$ is almost constant (equal to m) for a m-interlacing; it almost has a binomial distribution for an m-mixture; see also Exercise 12. Again, simulations are useful to decide which model provides the best fit.
- Size of connected components: An interesting problem is to identify the connected components in the undirected graph of nearest neighbors associated to a point process, see Exercise 20. These connected components are featured in Figure 2. Their size distribution is of particular interest: for instance, on the left plot in Figure 2, corresponding to $s = 0$, there is only one connected component of infinite size; on the right plot, there are infinitely many small connected components (about 50% only have two points). It is still an open question as to whether or not this statistic can be used to discriminate between different types of point processes, or whether its theoretical distribution is exactly the same for a large class of point processes (that is, it is an attractor distribution) and thus of little practical value.

Below I discuss a statistical test that I used many times, to check how different a set of observed points is, compared to one arising from a simple two-dimensional Poisson-binomial point process, or from a stationary Poisson point process, or more generally from any kind of stochastic point process.

Rayleigh Test

The Rayleigh test is a generic statistical test to assess whether two data sets consisting of points in two dimensions, arise from the same type of stochastic point process. It assumes that the underlying point process model is uniquely characterized by the distribution of nearest neighbor distances. The most popular use is when the assumed model is a stationary Poisson process: in that case, the statistic of the test has a Rayleigh distribution. It generalizes to higher dimensions; in that case the Rayleigh distribution becomes a Weibull distribution. In short, what the test actually does, is comparing the empirical distributions of nearest neighbor distances computed on the two datasets, possibly after standardization, to assess if from a statistical point of view, they are indistinguishable.

The test is performed as follows. Let's say you have two data sets consisting of points in two dimensions, observed through a window. You compute the empirical distribution of the nearest neighbor distances for both datasets, based on the observations, after taking care of boundary effects. Let $\eta_1(u)$ and $\eta_2(u)$ be the two distributions in question. The statistic of the test is

$$V = \int_{-\infty}^{\infty} |\eta_1(u) - \eta_2(u)| du = \int_0^1 |\nu_1(u) - \nu_2(u)| du, \tag{32}$$

where ν is the empirical quantile function, that is, the inverse of the empirical distribution. An alternative test is based on $W = \sup_u |\eta_1(u) - \eta_2(u)|$, or on $W' = \sup_u |\nu_1(u) - \nu_2(u)|$. The test based on W is the traditional Kolomogorov-Smirnov test [Wiki] with known tabulated values. In Excel, it is easier to use the empirical quantile function, readily available as the `PERCENTILE` Excel function. In practice, the integral in Formula (32) is replaced by a sum computed over 100 equally spaced value of $u \in [0, 1]$. The advantage of W is that it is known (asymptotically) not to depend on the underlying (possibly unknown) point process model that the data originates from.

I provide an illustration in `PB_inference.xlsx`: see the "Rayleigh test" tab in the spreadsheet. I compare two data sets, one from a simulation of a two-dimensional Poisson-binomial process with $s = 20$, and one with

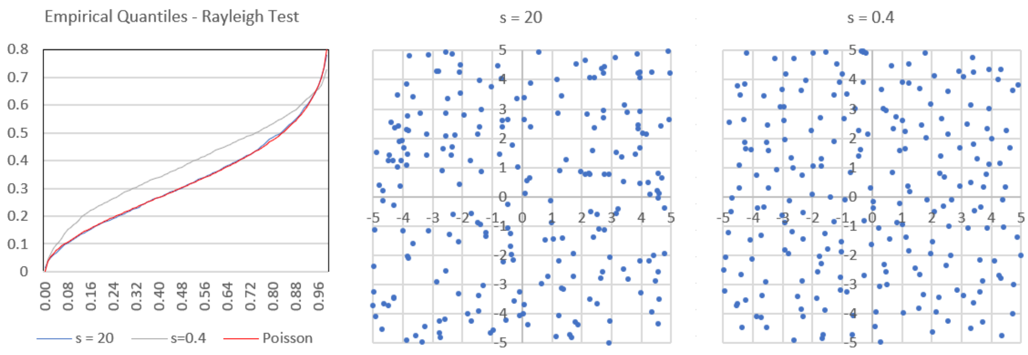

Figure 18: Rayleigh test to assess if a point distribution matches that of a Poisson process

$s = 0.4$. In both cases, λ is set to 1.5 in the simulator; its estimated value on the generated data set is close to 1.5. I then compare the nearest neighbor distances (their empirical quantile function) with the theoretical distribution of a two-dimensional stationary Poisson process of intensity λ^2. The theoretical distribution is Rayleigh of expectation $1/(2\lambda)$. The dataset with $s = 20$ is indistinguishable, at least using the Rayleigh test, from a realization of a stationary Poisson process. This was expected: as $s \to \infty$, the Poisson-binomial process converges to a Poisson process by virtue of Theorem 4.5, and the convergence is very fast. But the data set with $s = 0.4$ is markedly different from a Poisson point process realization, as seen by looking at the statistic V or W'.

Tabulated values for the statistics V and W' can be obtained by simulations. For W, they have been known since at least 1948, since W is the Kolomogorov-Smirnov statistic [26]. Here I simply used tabulated values of the Rayleigh distribution since I was comparing the simulated data with a realization of stationary Poisson process. Confidence bands [Wiki] for the empirical quantile function can be obtained using resampling methods [Wiki]. Modern resampling methods are discussed in details in my book "Statistics: New Foundations, Toolbox, and Machine Learning Recipes" [37] available here; see the chapters "Model-free, Assumption-free Confidence Intervals" and "Modern Resampling Techniques for Machine Learning". See also Section 3.1 in this textbook.

Figure 18 illustrates the result of my test, using the empirical quantile function of the nearest neighbor distances, and the statistic V for the test. No re-sampling or confidence bands were needed, the conclusion is obvious: $s = 0.4$ provides a simulated data set markedly different from a Poisson point process realization (the gray curve is way off) while $s = 20$ is indistinguishable from a Poisson point process (the red and blue curves, representing the empirical quantile function of the nearest neighbor distances, are almost identical). Interestingly, the scatterplot corresponding to $s = 0.4$ (rightmost in Figure 18) seems more random than with $s = 20$ (middle plot), but actually, the opposite is true. The plot with $s = 0.4$ corresponds to a repulsive process, where points are more away from each other than pure chance would dictate; thus it exhibits fewer big empty spaces and less clustering, falsely giving the impression of increased randomness.

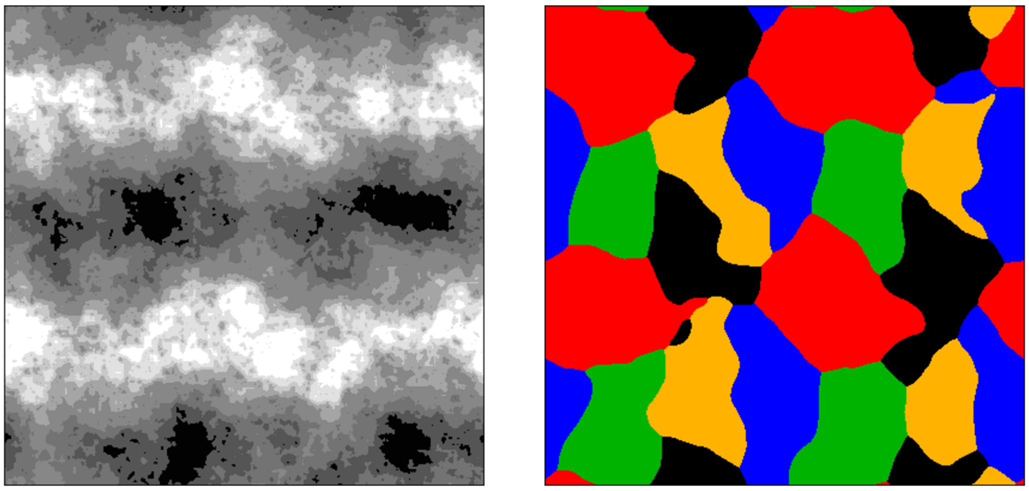
Figure 19: Unsupervised (left) versus supervised clustering (right) of Figure 17

3.4.3 Clustering Using GPU-based Image Filtering

In this section, I describe a methodology for very fast supervised and unsupervised clustering. The data is first transformed into a 400×400 two-dimensional array called *bitmap*. The points are referred to as pixels, and the array represents an image stored in GPU (the graphics processing unit) [Wiki]. The functions applied to the bitmap are standard image processing techniques such as high pass filtering or histogram equalization [Wiki]. The easy-to-read source code is in Section 6.6.2; it is accompanied by detailed comments about the methodology. I encourage you to read it.

The input data consists of a realization (obtained by simulation) of an m-interlacing (that is, a superimposition of m shifted Poisson-binomial processes) with each individual process represented by a different color: see Figure 17. The left plot in Figure 17 shows the data points observed through a small window $B = [-10, 10] \times [-10, 10]$. The right plot corresponds to a much bigger window, with all points taken modulo $2/\lambda$. So, despite the bigger window, the point locations, after the modulo operation, are in $[0, 2/\lambda] \times [0, 2/\lambda]$. I chose $\lambda = 1$ for the intensity, in the simulations. The modulo operation (see Section 3.4.1) magnifies the cluster structure, invisible on the left plot, and visible on the right plot.

The end result is displayed in Figure 19. The left plot corresponds to unsupervised clustering, including locating the shift vectors attached to each individual process of the m-mixture. The right plot corresponds to supervised clustering of the entire state space: the color of a point represents the individual point process it belongs to; in this case the data set is the training set.

Remark: For the simulations, see source code `PB_NN.py` in Section 6.4 (Part 2), or Formulas (8) and (9); m-mixtures are described in Exercise 18 and Sections 1.5.3, 1.5.4 and 3.4. See [29] (available online, here) for a similar use of GPU in the context of nearest neighbor clustering.

Supervised Clustering with High Pass Filter

Here the data set represents the training set. The algorithm consists of filtering the whole 400×400 bitmap 3 times. Each time, a local filter is applied to each pixel (x, y). Initially, the color $c(x, y)$ attached to the pixel represents the cluster it belongs to, in the training set (or in other words, the individual point process it originates from in the m-mixture): its value is an integer between 0 and $m - 1$ if it is in the training set, and 255 otherwise. The new color assigned to (x, y) is

$$c'(x, y) = \arg\max_j \sum_{u=-20}^{20} \sum_{v=-20}^{20} \frac{\chi[c(x-u, y-v) = j]}{\sqrt{1 + u^2 + v^2}}, \tag{33}$$

where $\arg\max g(j)$ [Wiki] is the value of j that maximizes $g(j)$, and $\chi[A]$ is the indicator function [Wiki]: $\chi[A] = 1$ if A is true, and 0 otherwise. The boundary problem (when $x - u$ or $y - v$ is outside the bitmap) is handled in the source code. Have a look at my solution (Part 2 of source code in Section 6.6.2), though there are many other ways to handle it.

After filtering the whole bitmap 3 times, thanks to the large size of the filtering window (21×21 pixels), all pixels are assigned to a cluster (a color different from 255). This means that any future point (not in the training set) can easily and efficiently be classified: first, find its location on the bitmap; then its cluster is the color assigned to that location. It is worth asking whether convergence occurs (and to what solution) if you were to filter the bitmap many times. I have not investigated this problem, however, I studied convergence for a similar type of filter, in my paper "Simulated Annealing: A Proof of Convergence" [38].

While the algorithm is very fast, the bottleneck is the large size of the local filter window. The amount of time required to color the bitmap is proportional to the size of that window: in our case, 21×21 pixels. There is a way to accelerate this by a factor about 20, using a caching mechanism. See Exercise 26.

Connection to Neural Networks

The filtering system is essentially a neural network [Wiki]. The image before the first iteration (Figure 17, right plot), consisting of the training set, is the input layer. The final image obtained after 3 iterations (Figure 19, right plot) is the output layer. The intermediate iterations correspond to the hidden layers. In each layer, a pixel color is a function of quantities computed on neighbor pixels, in the previous layer. This is a classic example of neural network! See Luuk Spreeuwers' PhD thesis "Image Filtering with Neural Networks" defended in 1992 [72] (available online, here), for more about image filters used as neural networks.

The pre-processing step consists of transforming the data set into a bitmap. In the next section about unsupervised clustering, the post-processing step called "equalizer" plays the role of the sigmoid function in neural networks.

Unsupervised Clustering with Density Equalization

A similar filter is used for unsupervised clustering. Much of what I wrote for unsupervised clustering also applies here. I recommend that you first read the above section about supervised clustering. Indeed, both supervised and unsupervised clustering are implemented in parallel in the source code, within the same loop. The main difference is that the color (or cluster) $c(x, y)$ attached to a pixel (x, y) is not known. Instead of colors, I use gray levels representing the density of points at any location on the bitmap: the darkest, the higher the density. I start with a bitmap where $c(x, y) = 1$ if (x, y) corresponds to the location of an observed point on the bitmap, and $c(x, y) = 0$ otherwise. Again, I filter the whole 400×400 bitmap 3 times with the same 20×20 filter size. The new gray level assigned to pixel (x, y) at iteration t is now

$$c'(x, y) = \arg\max_j \sum_{u=-20}^{20} \sum_{v=-20}^{20} \frac{c(x-u, y-v) \cdot 10^{-t}}{\sqrt{1+u^2+v^2}}. \tag{34}$$

The first time this filter is applied to the whole bitmap, I use $t = 0$ in Formula (34); the second time I use $t = 1$, and the third time I use $t = 2$. The purpose is to dampen the effect of successive filtering, otherwise the image (left plot in Figure 19) would turn almost black everywhere after a few iterations, making it impossible to visualize the cluster structure. The second and third iterations, with the dampening factor, provide an improvement over using a single iteration only.

After filtering the image, I applied a final post-processing step to enhance the gray levels: see Part 4 of the source code in Section 6.6.2. It is a purely cosmetic step consisting in binning and rescaling the histogram of gray levels to make the image nicer and easier to interpret. This step, called equalization, can be automated; I will discuss it in details in an upcoming textbook. I chose a data set with significant overlap among the clusters to show the power of the methodology. Indeed, if you look at the raw data (Figure 17, left plot), the cluster structure is invisible to the naked eye. This algorithm was able to only partially recover the cluster structure. The centers of the clusters visible in Figure 19 (left plot) roughly correspond to some of the shift vectors attached to the m-mixture. Retrieving the shift vectors was one of the goals.

The dataset used here is produced by the program `PB_NN.py` in Section 6.4. You can download the dataset from my GitHub repository, here. The first column is the cluster number: an integer $i \in \{0, \dots, m-1\}$ with $m = 5$; the fourth column is the X coordinate, and column $5 + i$ is the Y coordinate. To produce the images and manipulate the palettes, I used the Pillow graphics library. See Section 6.6.2.

3.4.4 Black-box Elbow Rule to Detect Outliers and Number of Clusters

In the context of unsupervised clustering, one of the most popular recipes to identify the number of clusters, is the elbow rule [Wiki]. It is usually performed manually. Here, I show how it can be automated and applied to other problems, such as outlier detection. The idea is a follows: a clustering algorithm (say k-means [Wiki]) can identify a cluster structure with any number of clusters on a given data set; typically, a function $v(m)$ provides a statistical summary of the best cluster structure consisting of m clusters, for $m = 1, 2, 3$ and so on. For instance, $v(m)$ is the sum of the squares of the distances from any observed point to its assigned cluster center. The function $v(m)$ is decreasing, sharply initially for small values of m, then much more slowly for larger values of m, creating an elbow in its graph. The value of m corresponding to the elbow is deemed to be the optimal number of clusters. See Figure 20. Instead of $v(m)$, I use the standardized version $v'(m) = v(m)/v(1)$.

Brownian Motions and Clustered Lévy Flights

I illustrate how to use the elbow rule to detect outliers in the next subsection. The same methodology applies to detect the number of clusters. First, let me introduce a new type of point process: the Brownian motion [Wiki], also called Wiener process. This type of process will be studied in more details in Volume 2 of this textbook. In one dimension, we start with $X_0 = 0$ and $X_k = X_{k-1} + R_k\theta_k$, for $k = 1, 2$ and so on. If the R_k's are independently and identically distributed (iid) with an exponential distribution of expectation $1/\lambda$ and $\theta_k = 1$, then the resulting process is a stationary Poisson point process of intensity λ on $\mathbb{R}^+$; the R_k's are the successive interarrival times. If the θ_k's are iid with $P(\theta_k = 1) = P(\theta_k = -1) = \frac{1}{2}$, and independent from the R_k's, then we get a totally different type of process, which, after proper re-scaling, represents a time-continuous Brownian motion in one dimension. I generalize it to two dimensions, as follows. Start with $(X_0, Y_0) = (0, 0)$. Then generate the points (X_k, Y_k), with $k = 1, 2$ and so on, using the recursion

$$X_k = X_{k-1} + R_k \cos(2\pi\theta_k) \tag{35}$$

$$Y_k = Y_{k-1} + R_k \sin(2\pi\theta_k) \tag{36}$$

where θ_k is uniform on $[0, 1]$, and the radius R_k is generated using the formula

$$R_k = \frac{1}{\lambda}\Big(-\log(1-U_k)\Big)^{\gamma}, \tag{37}$$

where U_k is uniform on $[0, 1]$. Also, $\lambda > 0$, and the random variables U_k, θ_k are all independently distributed. If $\gamma > -1$, then $\text{E}[R_k] = \frac{1}{\lambda}\Gamma(1+\gamma)$ where Γ is the gamma function [Wiki]. In order to standardize the process, I use $\lambda = \Gamma(1+\gamma)$. Thus, $\text{E}[R_k] = 1$ and if $\gamma > -\frac{1}{2}$,

$$\text{Var}[R_k] = \frac{\Gamma(1+2\gamma)}{\Gamma^2(1+\gamma)} - 1.$$

We have the following cases:

- If $\gamma = 1$, then R_k has an exponential distribution.
- If $-1 < \gamma < 0$, then R_k has a Fréchet distribution. If in addition, $\gamma > -\frac{1}{2}$, then its variance is finite.
- If $\gamma > 0$, then R_k has a Weibull distribution, with finite variance.

Interestingly, the Fréchet and Weibull distributions are two of the three attractor distributions in extreme value theory. In my opinion, Fréchet and Weibull should not be considered as two different families of distributions. See Section 3.5.2 for more details.

The two-dimensional process consisting of the points (X_k, Y_k) is a particular type of random walk [Wiki]. The random variables R_k represent the (variable) lengths of the successive increments. Under proper rescaling, assuming the variance of R_k is finite, it tends to a time-continuous two-dimensional Brownian motion. However, if $\text{Var}[R_k] = \infty$, it may not converge to a Brownian motion. Instead, it is very similar to a Lévy flight, and produces a strong cluster structure, with well separated clusters when the number of points is finite, see Figure 20. The Lévy flight uses a Lévy distribution [Wiki] for R_k, which also has infinite expectation and variance. Along with Cauchy (also with infinite expectation and variance), it is one of the stable distributions [Wiki]. Such distributions are attractors for an adapted version of the Central Limit Theorem (CLT), just like the Gaussian distribution is the attractor for the CLT. A well written, seminal book on the topic, is "Limit Distributions for Sums of Independent Random Variables", by Gnedenko and Kolmogorov [31].

For a simple introduction to Brownian and related processes, see the website RandomServices.org by Kyle Siegrist, especially the chapter on standard Brownian motions, here. My book "Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems" [36] (available online here), offers a fresh perspective and discusses original topics related to dynamical systems, all without any reference to measure theory, and thus accessible to beginners.

Elbow Rule to Detect Outliers

Figure 20 shows a realization of a Brownian motion with 10^4 points, using $\gamma = 2$ and $\lambda = \Gamma(1+\gamma)$ in Formula (37). The goal is to detect the number of values, among the top R_k's, that significantly outshine all the others. Here, they are not technically outliers in the sense that they are still deviates of the same distribution; rather, they are called extremes. The first step is to rank these values. The ordered values (in reverse order) are denoted as $R_{(1)}, R_{(2)}$ and so on, with $R_{(1)}$ being the largest one. I used $v(m) = R_{(m)}$ as the criterion for the elbow rule, that is, after standardization, $v'(m) = v(m)/v(1)$.

On the right plot in Figure 20, the Y axis on the left represents $v'(m)$, the X axis represents m, and the Y axis on the right represents the strength of the elbow signal (the height of the red bar; I discuss later how it is computed). The top 10 values of $v'(m)$ $(m = 1, \dots, 10)$ are

$$1.00, \quad 0.92, \quad 0.77, \quad 0.76, \quad 0.71, \quad 0.69, \quad 0.63, \quad 0.61, \quad 0.60, \quad 0.56, \quad 0.55, \quad 0.55.$$

Clearly, the third value 0.77 is pivotal, as the next ones stop dropping sharply, after an initial big drop at the beginning of the sequence. So the "elbow signal" is strongest at $m = 3$, and the conclusion is that the first two values $(2 = m - 1)$ outshine all the other ones. The purpose of the black-box elbow rule algorithm, is to automate the decision process: in this case deciding that the optimum is $m = 3$.

Note that in some instances, it is not obvious to detect an elbow, and there may be none. In my example, the elbow signal is very strong, because I chose a rather large value $\gamma = 2$ in Formula 37, causing the Brownian process to exhibit an unusually strong cluster structure, and large disparities among the top $v(m)$'s. A larger γ would generate even stronger disparities. A negative value of γ, say $\gamma = -0.75$, also causes strong disparities, well separated clusters, and an easy-to-detect elbow. The resulting process is not even Brownian anymore if $\gamma = -0.75$, since in that case, $\text{Var}[R_k] = \infty$. The standard Brownian motion corresponds to $\gamma = 0$ and can still exhibit clusters depending on the realization. Finally, in our case, $m = 3$ also corresponds to the number of clusters on the left plot in Figure 20. This is a coincidence, one that happens very frequently, because the top $v(m)$'s (left to the elbow) correspond to unusually large values of R_k. Each of these very large values typically gives rise to the building of a new cluster, in the simulations.

The elbow rule can be used recursively, first to detect the number of "main" clusters in the data set, then to detect the number of sub-clusters within each cluster. The strength of the signal (the height of the red bar) is typically very low if the $v'(m)$'s have a low variance. In that case, there is no set of values outshining all the other ones, that is, no true elbow. For an application of this methodology to detect the number of clusters, see a recent article of Chikumbo [14], available online here. An alternative to the elbow rule, to detect the number of clusters, is the silhouette method [Wiki].

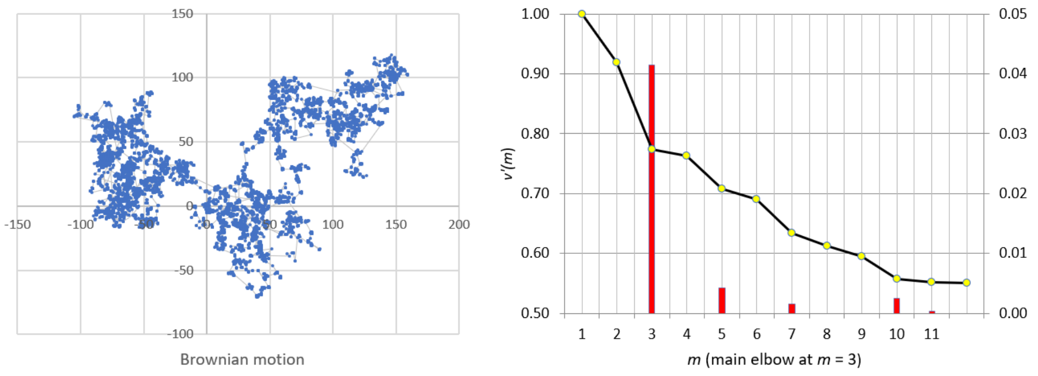

Figure 20: Elbow rule (right) finds $m = 3$ clusters in Brownian motion (left)

I now explain how the strength of the elbow signal (the height of the red bars in Figure 20) is computed. First, compute the first and second order differences of the function $v'(m)$: $\delta_1(m) = v'(m-1) - v'(m)$ for $m > 1$, and $\delta_2(m) = \delta_1(m-1) - \delta_1(m)$ for $m > 2$. The strength of the elbow signal, at position $m > 1$, is $\rho_1(m) = \max[0, \delta_2(m+1) - \delta_1(m+1)]$. I used a dampened version of $\rho_1(m)$, namely $\rho_2(m) = \rho_1(m)/m$, to favor cluster structures with few large clusters, over many smaller clusters. Larger clusters can always be broken down into multiple clusters, using the same clustering algorithm. The data, including formulas, charts, and simulation of the Brownian motion (done in Excel!), is on the `Elbow_Brownian` tab, in the `PB_inference.xls` spreadsheet. You can modify the parameters highlighted in orange in the spreadsheet: in this case, γ in cell `B16`. Note that λ is set to $\Gamma(1 + \gamma)$ in cell `B17`.

Back to the Riemann Zeta Function

Here I revisit the Riemann zeta function [Wiki] first explored in Section 2.3.2. I investigate a related deterministic sequence (X_k, Y_k) also starting at $(X_0, Y_0) = (0, 0)$, exhibiting some amount of chaos, as many sequences do in dynamical systems [Wiki]. This sequence, unlike that produced by Formulas (35),(36) and (37), converges, albeit chaotically. To the contrary, the Brownian motion sequence, at least when $\text{Var}[R_k]$ is finite, eventually covers the entire plane when the number of points is infinite, producing a dense plot that has a fractal dimension [Wiki]. The purpose here is to determine the number of "jumps" in the deterministic sequence from starting point to convergence, using the elbow rule. The deterministic sequence represents the partial sums of the Dirichlet eta function $\eta(z)$ [Wiki] in the complex plane, defined by Formulas (18) and (19). It is redefined here using Formulas (35) and (36), but this time with $\theta_k = t \log(k)$ and $R_k = (-1)^{k+1} k^{-\sigma}$, where $t > 0$ and $0 < \sigma < 1$ are two parameters: the imaginary and real part of the argument z of $\eta(z)$; in other words $z = \sigma + it$.

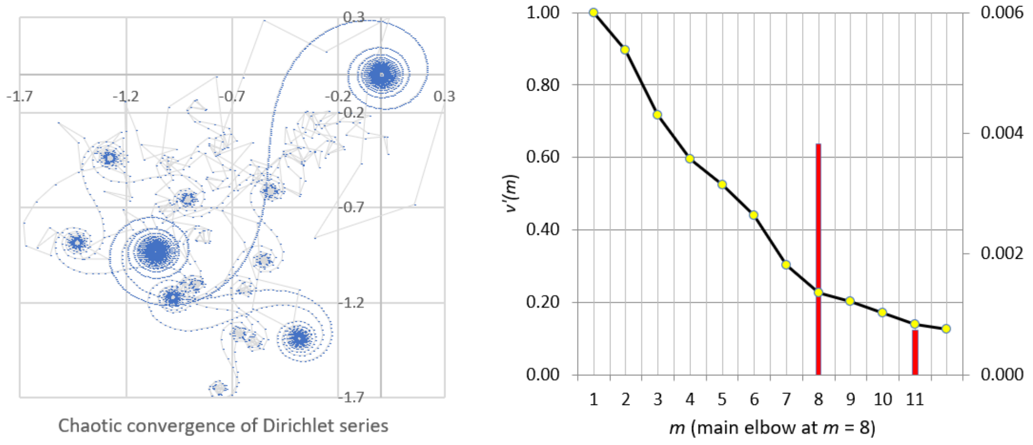

Figure 21: Elbow rule (right) finds $m = 8$ or $m = 11$ "jumps" in left plot

The left plot in Figure 21 represents the partial sums (X_k, Y_k) of $\eta(z)$ for the complex number $z = \sigma + it$, using the aforementioned formulas with k terms ($k = 1, \dots, 10^4$). The X axis represents the real part, the

Y axis the imaginary part. In complex number notation, (X_k, Y_k) is denoted as $X_k + iY_k$. Here $\sigma = \frac{1}{2}$ and $t = 24{,}556.59$. Not only this value of $\sigma + it$ is on the critical line [Wiki] since $\sigma = \frac{1}{2}$, but it is actually an excellent approximation to a non-trivial root [Wiki] of the Riemann zeta function. Thus, starting at $(0, 0)$, after an infinite number of steps ($k = \infty$), we end up back at $(0, 0)$ as shown on the left plot in Figure 21. In between, the path is pretty wild! No wonder why a proof of the famous Riemann Hypothesis [Wiki] still remains elusive.

I used the elbow rule to detect the number of sinks, denoted as m. A sink is when – on its path to convergence – the iterations get stuck for a while around a center, circling many times before resuming the normal path, creating the appearance of circular clusters. The final sink is centered at $(0, 0)$ since $\sigma + it$ is a root of η. If $\sigma > 0$ is close to zero, and t is large, the number of sinks can be much larger, and you may need far more than 10^4 iterations to reach the final sink, called "black hole". For the elbow rule, I first computed the empirical percentiles of the distance between (X_k, Y_k) and $(X_{k+\tau}, Y_{k+\tau})$ with $\tau = 100$, ignoring the first 1000 points where the path is most erratic. Then, I chose the $v(m)$'s as follows: $v(1)$ corresponds to the maximum distance, $v(2)$ to the 99-*th* percentile of the distances, $v(3)$ to the 98-*th* percentile, and so on. The remaining computations, once the $v(m)$ are computed, are identical to those in the previous section. The method found $m = 8$ sinks.

The data, including formulas, charts, and iterative computations of (X_k, Y_k) for $k = 1, \dots, 10^4$ (done in Excel), is on the `Elbow_Riemann` tab, in the `PB_inference.xls` spreadsheet. You can modify the parameters highlighted in orange in the spreadsheet: in this case, σ in cell `B16`, and t in cell `B17`. The reason why "jumps" appear in the sequence (X_k, Y_k) is explained and further illustrated for the one-dimensional case – the imaginary part of the Dirichlet eta function $\eta(z)$ – in Exercise 25. Tables of zeros of the Riemann zeta function (up to the first two million), published by Andrew Odlyzko, are available here.

3.5 Boundary Effect

A realization of a point process (or data), say in two dimensions, is typically observed through a rectangular window. Yet there are nearest neighbors located outside the observation window. Thus, there is a boundary effect, which may create a bias in statistics such as point count or nearest neighbor distance. The problem is obvious if you look at Figure 2. It is magnified if

- the underlying distribution F attached to each point of the Poisson-binomial process, has a thick tail (for instance, F is Cauchy),
- the number of observed points is small,
- or the scaling factor s is large.

The problem is mentioned throughout this textbook: see boundary effect in the glossary. In particular, see a solution in Section 3.1.2, in the context of parameter estimation.

In the literature, it is sometimes referred to as edge effect, see [5] available here. The unobserved data, outside the window of observations, is called censored data. Some statistics are more sensitive than others to boundary effects. The standard fix is to compute statistics of interest in a sub-window (best if you don't know the underlying model), or to correct for the bias (if you know the model).

For instance, say you simulate $(2N + 1)^2$ points (X_h, Y_k) with $h, k \in \{-N, \dots, N\}$. To minimize boundary effects when computing the average distance between nearest neighbors, you only look at points with index (h, k) satisfying $\max(|h|, |k|) \leq n$. Here, n is smaller than N. By how much? The topic of this section is to answer that question. Some nearest neighbors may satisfy $\max(|h|, |k|) \leq n$ and are thus in the smaller window (determined by n), and some may not but are still in the bigger window (determined by N). Some nearest neighbors could even be outside the bigger window if the difference between n and N is not big enough; in that case, a wrong nearest neighbor will be picked up, and a wrong nearest neighbor distance (too large for the point in question) will be computed. The end result is a bias in the average nearest neighbor distance computed on the observations, making it appear slightly larger than it actually is. By choosing N and n carefully, this problem can be minimized.

The issue is well illustrated in Figure 2: what you see is the smaller window; yet some arrows are pointing outside the window. These arrows point to nearest neighbors outside the small window; thus these neighbors have to be located in a bigger window, and indeed a bigger window (not shown in the picture) was used to produce the image.

3.5.1 Quantifying some Biases

Now, let's quantify the bias in question. If F has a thick tail or s is large, a point (X_h, Y_k) in the window of observations may have its index location (h, k) far away (that is, $\max(|h|, |k|) > N$), and thus won't be generated (in other words, missing). Also, some points, despite having an index location (h, k) with $\max(|h|, |k|) \leq n$ in the index space , might have their actual location (X_h, Y_k) in the state space, far outside the window in

F	n	λ	s	$N_1(n)$	$N_2(n)$	$N_3(n)$	$\rho(n)$
Logistic	100	1	5	38,712	1287	1689	3.2%
Logistic	100	1	1	39,814	186	589	0.5%
Logistic	50	1	5	9356	644	845	6.4%
Logistic	50	1	1	9907	93	294	0.9%
Uniform	100	1	5	39,600	400	801	1.0%
Uniform	100	1	1	40,000	0	401	0.0%
Uniform	50	1	5	9800	200	401	2.0%
Uniform	50	1	1	10,000	0	201	0.0%

Table 5: Bias due to boundary effect, for point count in 2-D

question. This point will be generated by the simulator, but may not be included in statistical estimations, and its creation is a waste of time. These are the two problems we face.

It turns out that some of the biases can be exactly computed, assuming you know the underlying model: in our case, a Poisson-binomial point process. Let $N = n$, $B(a_n) = [-a_n, a_n] \times [-a_n, a_n]$ be a square with $a_n > 0$ to be determined later, and $p_{h,k}(a_n) = P[(X_h, Y_k) \in B(a_n)]$. In two dimensions, we have:

$$p_{h,k}(a_n) = \Big[F\Big(\frac{a_n - h/\lambda}{s}\Big) - F\Big(\frac{-a_n - h/\lambda}{s}\Big)\Big] \times \Big[F\Big(\frac{a_n - k/\lambda}{s}\Big) - F\Big(\frac{-a_n - k/\lambda}{s}\Big)\Big].$$

Also, let $I_n = \{(h,k) \in \mathbb{Z}^2, \text{ with } \max(|h|,|k|) \leq n\}$, and

$$\begin{aligned}
N_1(n) &= \sum_{(h,k)\in I_n} p_{h,k}(a_n),\\
N_2(n) &= \sum_{(h,k)\notin I_n} p_{h,k}(a_n),\\
N_3(n) &= \sum_{(h,k)\in I_n} (1 - p_{h,k}(a_n)) = (2n+1)^2 - N_1(n).
\end{aligned}$$

The quantities $N_1(n), N_2(n), N_3(n)$ represent respectively the expected number of observed points in the small window $B(a_n)$, the expected number of missing (unobserved) points in the same window, and the expected number of points outside $B(a_n)$ that were generated by the simulator if $(h,k) \in I_n$. The bias, when counting the points in $B(a_n)$ generated by the simulator, is thus $N_2(n)$.

For a fixed n, it is possible to find a_n that minimizes $N_2(n)/N_1(n)$, but in practice, $a_n = n/\lambda$ is good enough. Table 5 shows the bias $N_2(n)$ obtained with $\lambda = 1$ and $a_n = n$. The ratio $\rho(n) = N_2(n)/(N_1(n) + N_2(n))$ is the proportion of bias. Assuming $\lambda = 1$, the unbiased point count (expected value) is $4n^2$; the biased count is $N_1(n)$.

I used the `CDF` function in Section 6.2.4 to compute the statistics N_1, N_2 and N_3 in Table 5. The source code, illustrating the use of a bivariate cumulative distribution function F, is as follows:

```
N1=0
N2=0
N=2000 # should be infinite, but 2000 is good enough
n=100
llambda=1
s=5
aa=n # aa corresponds to a_n in the text
type="Logistic"

for h in range(-N,N+1):
  print(h)
  for k in range(-N,N+1):
   ff=(CDF(type,llambda,s,h,aa)-CDF(type,llambda,s,h,-aa)) \
            * (CDF(type,llambda,s,k,aa)-CDF(type,llambda,s,k,-aa))
   if abs(k)<=n and abs(k)<=n:
     N1+=ff
   else:
     N2+=ff
N3=(2*n+1)*(2*n+1)-N1
print("N1=",int(N1),"N3=",int(N3))
```

For a fixed, large n, as s increases, both $N_2(n)$ and $N_3(n)$ increase, but their ratio tends to 1 as $s \to \infty$. This is because as $s \to \infty$, the Poisson-binomial process tends to a stationary Poisson process.

Remark: If $a_n = n/\lambda$, by virtue of Theorem 4.1 generalized to two dimensions, $N_1(n)+N_2(n) = \lambda^2\mu(B(a_n)) = (2n)^2$.

3.5.2 Extreme Values

Figure 22 shows the points (X_h, Y_k) of a Poisson-binomial process with a logistic F, in the state space (blue dots) and their index location $(h/\lambda, k/\lambda)$ in the lattice space (red crosses), connected by arrows. Here, $\lambda = 1$, thus the index space and the lattice space are identical. The source code to produce Figure 22 is provided in Section 6.6.1.

This picture shows how far away a point can be from the lattice location it is attached to. If $s = 0$, both locations coincide, but when s is large, that is, when points are distributed as in a stationary Poisson process, the distance between the point and its lattice location can be very large, for most points. A large s magnifies the boundary effects when performing simulations. Note that both plots (left and right in Figure 22) have the same number of points. But points are clustered in some areas, and sparse in other areas on the right plot, giving the impression that there are fewer of them. Clearly, the empirical distribution of the distance between nearest neighbors (especially extreme distances), or the average area of the largest empty zone, can be used to estimate the scaling factor s once λ is known or estimated.

This brings me to my next discussion: extreme values, or records. This is part of a field know as order statistics [Wiki] or extreme value theory [Wiki]. Extreme values are different from outliers [Wiki]: they can be predictable, with known distribution. To the contrary, outliers are usually considered as errors, glitches, or data points obeying a different model. In any case, both have an impact on the window of observations, delimited by the "boundary", and have the potential to introduce biases.

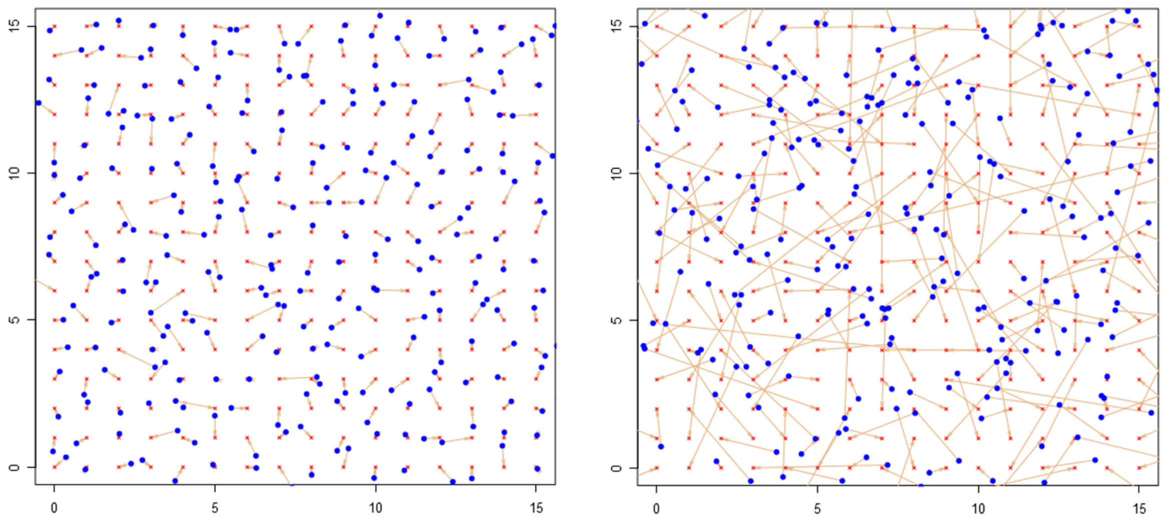

Figure 22: Each arrow links a point (blue) to its lattice index (red): $s = 0.2$ (left), $s = 1$ (right)

One question is how far a point can be from its lattice location, and how frequently such "extremes" occur. Even more interesting is the reverse question, associated to the inverse or hidden model: can a point (X_h, Y_k) close to the origin, well within the small window of observations, have its lattice location (h, k) very far away? Such a point will not be generated by the point process simulator. It will be unaccounted for, introducing a bias; indeed, it is counted in $N_2(n)$. This happens with increased frequency as s increases, requiring a larger and larger observation window (that is, larger n and N), as seen in Table 5.

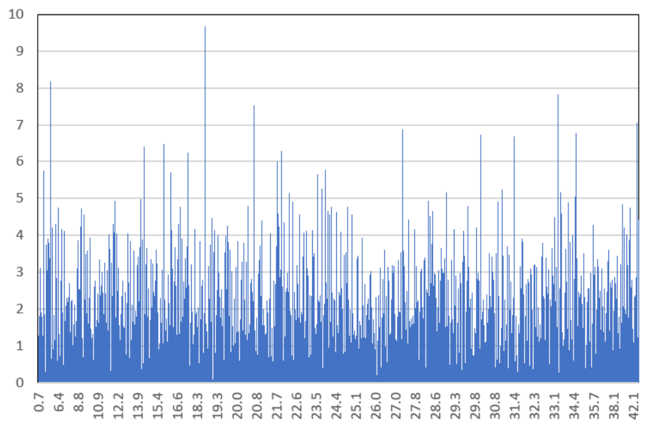

Figure 23: Distance between a point and its lattice location ($s = 1$)

Unless F has a finite support domain (for instance, if F is uniform), unobserved points in the small window of observations – even though their expected number is finite and rather small – can be attached to any arbitrary lattice location, not matter how far away. In two dimensions, the probability $P[R > r]$ that the distance R between a point and its lattice location is greater than r, is

$$P(R > r) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \chi(x^2 + y^2 > r)F\Big(\frac{x}{s}\Big)F\Big(\frac{y}{s}\Big)dxdy$$

where $\chi(A)$ is the indicator function, equal to one if A is true, and to zero otherwise.

The distance R corresponds to the length of the arrow, in Figure 22. If F is Gaussian, then R has a Rayleigh distribution [Wiki]. In two dimensions, the distance between two nearest neighbor points, for a stationary Poisson point process, also has a Rayleigh distribution, see Section 3.4 and Exercise 15.

Distribution of Records

Now let M_n be the maximum distance between a point and its lattice location, measured over n points of the process, randomly selected. In other words $M_n = \max(R_1, \dots, R_n)$ where R_i ($i = 1, \dots, n$) is the distance between the i-th point, and its lattice location. Depending on F, the standardized distribution of M_n is asymptotically Weibull, Gumbel or Fréchet: these are the tree potential attractor distributions in the context of extreme value theory [Wiki]. The Rayleigh distribution is a particular case of the Weibull distribution. Surprisingly, in d dimensions, the distribution of the nearest neighbor distances, for a stationary Poisson point process, is also Weibull, see Section 3.4.

Figure 23 shows (on the Y-axis) the distance R between a point (X_h, Y_k) and its location $(h/\lambda, k/\lambda)$ on the lattice space. These are the same points as on the right plot in Figure 22; R represents the length of the arrows. The points are ordered by how close they are to the origin $(0, 0)$, and the X-axis represents their distance to the origin, that is, their norm. By looking at Figure 23, it is easy to visualize the extreme values of R, and when they occur on the X-axis.

Distribution of Arrival Times for Records

Now let us assume that n is infinite, and let's look at the arrival times of the successive records in the sequence R_1, R_2, R_3 and so on. The i-th arrival time is denoted as L_i with $L_1 = 1$, and defined as follows: $L_{i+1} = \min\{j : R_j > R_{L_i}\}$. In other words, the i-th record is R_{L_i}. The random variable L_i has the following properties:

- The distribution of L_i does not depend on F.
- Let η_i be the probability that R_i is a record. The η_i's are independent Bernoulli random variables, and $P(\eta_i = 1) = 1/i$.
- $P(L_i \geq m) = P(\eta_1 + \eta_2 + \cdots + \eta_m \leq i)$. We are again dealing with a Poisson-binomial distribution [Wiki].
- $\mathrm{E}[L_i] = \infty$ if $i > 1$. However, $\mathrm{E}[\log L_i] \sim i - \gamma$ as $i \to \infty$, where $\gamma = 0.5772\dots$ is the Euler–Mascheroni constant [Wiki].
- $\mathrm{Var}[\log L_i] \sim i - \pi^2/6$ as $i \to \infty$.

These results, and many others, are found in chapter 19 (*A Record of Records*) in Balakrishnan handbook entitled "Order Statistics: Theory & Methods" [7]. See pages 517–525.

3.6 Poor Random Numbers and Other Glitches

All machine learning and modeling techniques are subject to a number of issues. I discussed the boundary effect in Section 3.5, creating biases in some statistical measurements, and how to address it. Perturbed lattice point processes, referred to as Poisson-binomial processes in this textbook, are unusually stable structures. However on occasions, one may face numerical stability or precision issues. For instance, the detection of connected components (those generated by the nearest neighbors) can fail if the scaling factor s is zero. In that case, a point can have multiple nearest neighbors, causing problems. This is addressed in Part 3 of the source code in Section 6.4. Another example is caused by the chaotic convergence of some mathematical series: see Exercise 25, with a solution. Limiting distributions near a singularity are another typical source of problems, see Exercise 4, entitled *small paradox*. Iterative algorithms such as the filter-based classifier in Section 3.4, used to produce Figure 19, may not converge or converge to a wrong solution depending on the parameters.

But generally speaking, iterative systems going awry are rare when dealing with lattice-based point processes. This is in contrast to discrete dynamical systems , where a simple recursion such as $x_{n+1} = 4x_n(1 - x_n)$ with $0 < x_0 < 1$ (called the chaotic logistic map) yields erroneous values with not a single correct digit after as little as $n = 50$ iterations, when using single-precision arithmetic [Wiki]. This is not an issue to compute average-based statistics due to the ergodicity of the dynamical system, mimicking a stochastic process. It becomes an issue when looking at a single path, or when computing statistics such as long-range auto-correlations to assess the randomness of the sequence.

Surprisingly, in some instances, using a faulty algorithm can be a blessing. For instance, to find the global minimum of the chaotic curve pictured in Figure 7, standard optimization techniques such as the fixed point algorithm [Wiki], fail. Instead, I used a fixed point algorithm that by design, never converges. Yet as the iterations approach the (magnified) global minimum of the transformed function, it emits a signal before moving away to nowhere. It is possible to retrieve the global minimum via the signal. This will be discussed in an upcoming textbook.

In our context, since I heavily rely on massive simulations, in particular to estimate a number of theoretical distributions with good enough accuracy or to compare two very similar empirical distributions, an excellent pseudo-random number (PRNG) generator is paramount. Nowadays, most programming languages and even Excel offer decent PRNGs. See also here for a discussion on this topic. I have used billions of binary digits of peculiar transcendental numbers [Wiki] on many occasions: they provide some of the best non periodic PRNGs. You can get one million binary digits of (say) $\sqrt{2}$, online in less than one second, on the Sage symbolic math calculator, here. I now discuss a situation where my PRNG dramatically failed, and a new type of PRNG that I am currently developing.

3.6.1 A New Type of Pseudo-random Number Generator

The problem arose when I was performing simulations related to "six degrees of separation" [Wiki]. I needed to generate a few million IDs (each one representing an individual), and for each individual, randomly assign (say) 20 friends, then for each friend, another 20 friends and so on. The purpose was to find out whether there was a path between any two people, involving no more than six degrees of separation, and to estimate the average number of degrees of separation between two random people. The pseudo-random generator (PRNG) that I used was able to generate only 32,767 distinct numbers, and thus it miserably failed. So you might want to check if your PRGN has a similar issue.

Over the last 10 years, I have designed and tested many types of PRGN for cryptographic applications. The most interesting ones will be discussed in an upcoming textbook. Here I discuss the most recent one (still a work in progress) as it is simple and related to the material discussed in the subsection "chance of detecting large factors in very large integers", in Section 2.3.1. Its sequence is defined as follows: $y_n = \text{mod}(x_n, c)$ with $c = 2$ and

$$x_{n+1} = x_n + \sum_{k=1}^{r} a_k \bmod (x_n, p_k) + \sum_{k=1}^{r} b_k \bmod (n+1, p_k).$$

The initial value is x_1, a positive integer; p_1, p_2 and so on are the prime numbers, with $p_1 = 2$. Also, $a_k, b_k \in \{-1, 0, 1\}$. The sequence is periodic, though the period may start after a large number of iterations. In general, the larger r, the larger the period. This PRNG is further discussed here.

The parameter set in Table 6 yields a period equal to 643,032,390 $= 2 \times 3 \times 5 \times 7^3 \times 11 \times 13 \times 19 \times 23$. Detecting the period of these PRNG's, either via an algorithm or through theoretical considerations, is an interesting problem in and of itself. The period grows exponentially fast with the number of prime numbers involved. The number of iterations before the period starts to kick in can be very large. This makes it difficult to detect the period. But to make things easier, the period typically has a simple form, involving the product of consecutive primes. So one can try an integer q (a simple product of primes) and check if for some n large

k	1	2	3	4	5	6	7	8	9
p_k	2	3	5	7	11	13	17	19	23
a_k	-1	1	1	1	-1	-1	0	1	-1
b_k	-1	-1	-1	-1	1	1	0	-1	1

Table 6: Parameter set used in pseudo-random number generator

enough, $x_n = x_{n+q}, x_{n+1} = x_{n+q+1}, \dots, x_{n+200} = x_{n+q+200}$. If this is the case, q is a potential candidate for the period.

4 Theorems

The theorems presented here are selected for their practical and educational value. The proofs are usually short, constructive, and sometimes subtle. These results are used in one way or another throughout this textbook, including in the simulations. The reader is invited to try proving some of them on her own, before reading my solutions. I have added many comments, which are just as important as the theorems or the proofs. Emphasis is on making this material accessible to many practitioners as well as beginners, and hopefully, fun to read.

Remark: Unless otherwise specified, the theorems are valid for the one-dimensional case. Generalizations to higher dimensions are provided for several theorems, following the proof.

4.1 Notations

I use the notation $t_k = k/\lambda$ and $F_s(x) = F(x/s)$. The density attached to F, if it exists, is denoted as f. Also $B = [a, b]$ with $a < b$ is an interval on the real line. In two dimensions, B may be a rectangle or a circle, and I use the notation $\mu(B)$ for the area of B. The notation "Left $\equiv$ Right" means that "Left" is a shorter notation for "Right": by definition, they both represent the same thing.

The random variable $T(\lambda, s)$ measuring interarrival times is sometimes denoted as T. It represents the distance between two successive points of the process, once the points are ordered by value on the real line. In higher dimensions, T is the distance between a point of the process, and its closest neighbor. The random variable counting the number of points of the process in B is denoted as $N(B)$ and called point count.

4.2 Link between Interarrival Times and Point Count

The fundamental property linking N and T is as follows: $T > y$ (with $y \geq 0$) if and only if no points of the process are in the interval $B_0 =]X_0, X_0 + y]$, that is, if and only if $N(B_0) = 0$. Since X_0 is a random variable, this translates into the following.

$$\begin{aligned} P(T > y) &= \int_{-\infty}^{\infty} f(x) P(N(B_0) = 0 | X_0 = x) dx \\ &= \int_{-\infty}^{\infty} f(x) \prod_{k \neq 0} \Big(1 - P(X_k \in B_0)\Big) dx \\ &= \int_{-\infty}^{\infty} \frac{f(x)}{1 - p_0(x, y)} \prod_{k \in \mathbb{Z}} \Big(1 - p_k(x, y)\Big) dx \end{aligned} \tag{38}$$

where

$$p_k(x, y) = P(X_k \in]x, x + y]) = F_s(x + y - t_k) - F_s(x - t_k). \tag{39}$$

A different way to compute the distribution of the interarrival times is offered by Theorem 4.7. Note that T depends on λ and s, since $t_k = k/\lambda$. By analogy with the Poisson-binomial distribution attached to the counting random variable $N(B)$, the distribution of T is said to be exponential-binomial of parameters $p_k(x, y), k \in \mathbb{Z}$. When $s \to \infty$, the limit is a standard exponential distribution, as seen in Theorem 4.5.

I am now in a position to state and prove some important results. Unless otherwise specified, the theorems apply to the one-dimensional case. Following each proof, when possible, I discuss how the result generalizes to higher dimensions.

4.3 Point Count Arithmetic

Here is a pretty curious arithmetic-related result, easy to prove.

Theorem 4.1 *Regardless of the distribution F_s, if $\lambda \cdot (b-a)$ is an integer, then $E[N(B)] = \lambda \cdot \mu(B) = \lambda \cdot (b-a)$.*

Proof

For any function F_s, we have the following trivial equality. Assuming $\lambda \cdot (b-a) = 1$,

$$\mathrm{E}_n[B] \equiv \sum_{k=-n}^{n} \Big(F_s(b-t_k) - F_s(a-t_k) \Big) = F_s(b+t_n) - F(a-t_n).$$

Since F_s is a probability distribution, we have $F_s(b+t_n) \to 1$ and $F_s(a-t_n) \to 0$, thus $F(b+t_n) - F(a-t_n) \to 1$ as $n \to \infty$. Note that $1 = \lambda \cdot (b-a)$. A similar argument involving $2r$ terms on the right hand side, r of them that tend to 1, minus r of them that tend to 0 as $n \to \infty$, leads to the limit $(r \times 1) - (r \times 0) = r$ if $r = \lambda \cdot (b-a)$ is a positive integer. To conclude, note that $\mathrm{E}[N(B)] = \lim_{n\to\infty} \mathrm{E}_n[N(B)]$. ∎

By contrast, for a Poisson process, $\mathrm{E}[N(B)] = \lambda\mu(B)$ is always true. For Poisson-binomial processes, this is not the case, as illustrated in Theorem 4.8. This fact can be used to test whether you are dealing with a Poisson, or a Poisson-binomial process. For related results, see Section 3.1.2.

In two dimensions, the theorem generalizes as follows. Let $B = [a_1, b_1] \times [a_2, b_2]$ be a rectangle, and assume both $\lambda \cdot (b_1 - a_1)$ and $\lambda \cdot (b_2 - a_2)$ are integers. Then $\mathrm{E}[N(B)] = \lambda^2\mu(B) = \lambda^2(b_1 - a_1)(b_2 - a_2)$. In d dimensions $\lambda^2\mu(B)$ becomes $\lambda^d\mu(B)$, and $\mu(B)$ is the hypervolume of B.

4.4 Link between Intensity and Scaling Factor

The following theorem allows us to reduce the parameter space from two to one parameter.

Theorem 4.2 *Regardless of the distribution F_s, the interarrival times satisfy*

$$T(\lambda, s) = \frac{T(1, \lambda s)}{\lambda}.$$

In particular, this also holds when $s = \infty$, corresponding the standard Poisson process.

Proof

After replacing t_k by k/λ in (38), and since $F_s(z) = F(z/s)$, we have:

$$p_k(x, y) = F\Big(\frac{x + y - k/\lambda}{s}\Big) - F\Big(\frac{x - k/\lambda}{s}\Big).$$

The expression $F((x + y - k/\lambda)/s)$ can be rewritten as $F((\lambda \cdot (x+y) - k/\lambda')/s')$ with $\lambda' = 1$ and $s' = \lambda s$. This works too if $y = 0$. With the change of variable $\lambda \cdot (x + y) = x' + y$, we have $dx = (dx')/\lambda$ and the expression becomes $F((x' + y - k/\lambda')/s')$. The variables are x, x', and y is assumed to be fixed. Integral (38), after these changes, must be updated as follows:

- The dummy variable x is replaced by the dummy variable x'
- The value of the integral is divided by λ because $dx = (dx')/\lambda$
- The bounds are still from $-\infty$ to ∞
- λ is replaced by $\lambda' = 1$ and s by $s' = \lambda s$

That is: $P(T(\lambda, s) > y) = P(T(\lambda', s')/\lambda > y) = P(T(1, \lambda s)/\lambda > y)$, thus $T(\lambda, s) = T(1, \lambda s)/\lambda$ ∎

Theorem 4.2 has important practical implications. Instead of working with two parameters λ, s, when dealing with interarrival times, you can replace $T(\lambda, s)$ by $T^*(s') = \frac{1}{\lambda}T(1, s')$, with $s' = \lambda s$, thus reducing the number of effective parameters from two to one. I use this fact in Section 3.2.1 to facilitate estimation techniques, and to compute the empirical distribution [Wiki] of T more efficiently.

It would be interesting to see how Theorem 4.2 (and its proof) can be adapted to the two-dimensional case, where interarrival times are replaced by distances between a point of the process and its nearest neighbor. Simulations show that the situation is different. In two dimensions, x is replaced by (x_1, x_2), and dx becomes $dx_1 dx_2$. The product over k becomes a double product over h, k. Also, $F_s(x - k/\lambda)$ is replaced by $F_s(x_1 - h/\lambda)F_s(x_2 - k/\lambda)$, and $dx_1 = (dx'_1)/\lambda, dx_2 = (dx'_2)/\lambda$. This suggests that the denominator λ in Theorem 4.2 should be replaced by λ^2 in two dimensions. See also Exercise 15.

4.5 Expectation and Limit Distribution of Interarrival Times

Here I discuss the one-dimensional case. For the two-dimensional case, see Exercise 15. The proof of the next theorem justifies the choice of X_0 as the reference point to define interarrival times; X_5 (say) would have led to the same distribution. We already know that if $s = 0$ then $T = 1/\lambda$, and if $s = \infty$ then T has an exponential distribution of expectation $1/\lambda$. If s is small enough and F's tail is not too thick, then $\text{E}[T] = 1/\lambda$ and T's distribution is also independent from X_0, see Exercise 5. Now, the result below is valid for any $s \geq 0$.

Theorem 4.3 *If F has a finite expectation, then $\text{E}[T(\lambda, s)] = 1/\lambda$, regardless of F and s.*

Proof
Let (X_k) with $k = -n, \dots, n$ be a finite version of a Poisson-binomial point process, with $2n + 1$ points. One of the points, say X_{k_1}, is the minimum, and another one, say X_{k_2}, is the maximum. The range for the X_k's is $X_{k_2} - X_{k_1}$, with $\text{E}[X_{k_2}] = n/\lambda$ and $\text{E}[X_{k_1}] = -n/\lambda$. So the expectation of the range is $2n/\lambda$. Since there are $M = 2n$ interarrival times between X_{k_1} and X_{k_2}, the average interarrival time, that is the average distance between two successive points, is $\frac{1}{\lambda}\frac{2n}{M} = \frac{1}{\lambda}$. This is true whether n is finite or infinite. To finalize the proof, due to the symmetry of the problem (there is nothing special about X_0 versus, say, X_5), it does not matter, as far as the theoretical expectation is concerned, whether T is defined as the distance between X_0 and the next point to the right, or between X_5 (or any other point) and the next point. ■

If F is Cauchy, T's expectation may not exist. But in practice, we work with symmetric truncated Cauchy distributions [Wiki], that have zero expectation. Since the choice of the point X_0 does not matter in the definition of T, one might replace X_0 by the closest point to the origin. At least that point is known (observable) while X_0 is not. The next theorem, though surprisingly easy to prove, is much deeper than Theorem 4.3. I use it to solve Exercise 6.

Theorem 4.4 *If F has a density f, then*

$$\lim_{s\to 0} P\Big[\frac{1}{s}\Big(T(\lambda, s) - \frac{1}{\lambda}\Big) < y\Big] = \int_{-\infty}^{\infty} F(y - x)f(x)dx. \tag{40}$$

Proof
Note that $\text{E}[T(\lambda, s)] = \frac{1}{\lambda}$ by virtue of Theorem 4.3. When $s \to 0$, then $X_k \to \frac{k}{\lambda}$. It is then easy to establish (see Exercise 5) that

$$P(T < y) = \int_{-\infty}^{\infty} F\Big(x + \frac{y - 1/\lambda}{s}\Big)f(x)dx.$$

This can be rewritten as

$$P\Big[\frac{1}{s}\Big(T(\lambda, s) - \frac{1}{\lambda}\Big) < y\Big] = \int_{-\infty}^{\infty} F(y + x)f(x)dx = \int_{-\infty}^{\infty} F(y - x)f(x)dx.$$

The last equality is justified by the fact that f is symmetric, thus $f(x) = f(-x)$. The integral on the right hand side of Formula (40) represents the self-convolution [Wiki] of F. ■

4.6 Convergence to the Poisson Process

This theorem establishes, under mild conditions, the convergence to a Poisson process when $s \to \infty$.

Theorem 4.5 *If the distribution F has a density f, continuous almost everywhere, then the Poisson-binomial process converges to a stationary Poisson point process of intensity λ^d, as $s \to \infty$. Here d is the dimension of the state space.*

Proof
I proceed in two steps, to prove the result when $d = 1$.

Step 1

From (39), we have $p_k(x, y) = \int_a^b f(u)du$, where f (the density) is the derivative of F, $b = \frac{1}{\lambda s}(\lambda(x + y) - k)$, and $a = \frac{1}{\lambda s}(\lambda x - k)$. This integral has interval length $b - a = \frac{y}{s}$ and midpoint $\frac{1}{2}(a + b) = \frac{1}{2s}(2x + y) - \frac{k}{\lambda s}$. In particular,

$$p_k(x, y) \sim \frac{y}{s} f\Big(\frac{2x + y}{2s} - \frac{k}{\lambda s}\Big) \text{ as } s \to \infty,$$

$$J_n \equiv \sum_{k=-n}^{n} p_k(x, y) \sim \int_{-n}^{n} p_\nu(x, y)d\nu = \frac{y}{s}\int_{-n}^{n} f\Big(\frac{2x + y}{2s} - \frac{\nu}{\lambda s}\Big)d\nu.$$

With the change of variable $\tau = -\nu/(\lambda s)$, we obtain

$$J_n \sim \frac{y}{s} \cdot \Big[\lambda s \int_{-n/(\lambda s)}^{n/(\lambda s)} f\Big(\frac{2x+y}{2s} + \tau\Big) d\tau\Big] = \lambda y \int_{-n/(\lambda s)}^{n/(\lambda s)} f\Big(\frac{2x+y}{2s} + \tau\Big) d\tau.$$

Here λ is fixed. When $n \to \infty$, $s \to \infty$ and $n/s \to \infty$ (say $s \sim \sqrt{n}$ or $s \sim n/(\log n)$, we have

$$J_n \to \lambda y \int_{-\infty}^{\infty} f\Big(\frac{2x+y}{2s} + \tau\Big) d\tau = \lambda y,$$

because f is a density and thus integrates to one.

Step 2

Regardless of k, we have $p_k(x,y) \to 0$ as $s \to \infty$. So the denominator $1 - p_0(x,y)$ in (38) can be ignored when $s = \infty$. We also have:

$$\begin{aligned}\log\Big[\prod_{k\in\mathbb{Z}} \Big(1 - p_k(x,y)\Big)\Big] &= \sum_{k=-\infty}^{\infty} \log\Big(1 - p_k(x,y)\Big) \\ &\sim -\sum_{k=-\infty}^{\infty} p_k(x,y) \\ &= -J_\infty = -\lambda y.\end{aligned}$$

Thus,

$$\prod_{k\in\mathbb{Z}} \Big(1 - p_k(x,y)\Big) \sim \exp(-\lambda y) \text{ as } s \to \infty.$$

This product does not (at the limit) depend on x. Finally, we get

$$P(T > y) \sim \exp(-\lambda y) \int_{-\infty}^{\infty} \frac{f(x)}{1 - p_0(x,y)} dx \sim \int_{-\infty}^{\infty} f(x)dx = \exp(-\lambda y),$$

as f is a density and thus integrates to one. So, T has an exponential distribution of parameter λ as $s \to \infty$. This implies that the limiting point process must be Poisson of intensity λ. ∎

The takeaway from the proof of Theorem 4.5 (see bottom of Step 1) is that to simulate a realistic Poisson process as a limit of a Poisson-binomial process (pretty much regardless of F), you generate your $2n+1$ points (k between $-n$ and n), you choose a large n and a large s, but s must be an order of magnitude smaller than n, to make boundary effect [Wiki] negligible. For instance, $s = \sqrt{n}$ or $s = \frac{n}{\log n}$ will do.

Theorem 4.5 generalizes to higher dimensions. It is somewhat similar to the Central Limit Theorem [Wiki] (CLT), in the sense that it works regardless of the continuous distribution F. Even if the index space $\mathbb{Z}$ (the support domain for the index k) was relatively random, it would still work. What is remarkable is that even if F is a Cauchy distribution, known to have no expectation nor variance, it still works, and convergence to the Poisson process is even faster than if F was uniform. This is because the Cauchy distribution, with its thick tail, does a great job at mixing the points of the process. To the contrary, the standard CLT fails with a Cauchy distribution, as the sum of iid Cauchy random variables always has a Cauchy distribution, thus never converging to a Gaussian distribution. This is because the Cauchy distribution, like the Gaussian one, belongs to a family of stable distributions [Wiki].

In our case, convergence to a Poisson process is quite fast, with $s = 40$, assuming $\lambda = 1$, yielding an excellent approximation regardless of F, see Table 3. Consequently, the interest here is in small values of s. There might be a different way to prove Theorem 4.5, using Le Cam's inequality [73] applied to the point count distribution. It would amount to proving that as $n \to \infty$, regardless of B, the Poisson-binomial distribution of $N(B)$ tends to a Poisson distribution of expectation $\lambda^d \mu(B)$, where d is the dimension, and $\mu(B)$ is the area of B in two dimensions, or the length of the interval B in one dimension. See Theorem 2.1 for such a proof, in a similar context.

4.7 The Inverse or Hidden Model

Now I turn to what is usually referred to as the hidden model, as in hidden Markov models [Wiki]. Given an observed point x (a point of the process), what is the probability that $x = X_k$, for a specific k? In other words, can you retrieve the unknown index k attached to an observed point x? The related discrete random variable, indicating the index k attached to x, is denoted as $L(x)$, and takes on integer values in $\mathbb{Z}$. It is assumed that λ, s are known or estimated. Another random variable of interest, denoted as K and also taking on integer values (positive or negative) is is the index of the point closest to the point X_0, on the right-hand side on the real axis. Related material in the literature includes "Recovering the lattice from its random perturbations" by Yakir [79] (2020) available online here and "Cloaking the Underlying Long-Range Order of Randomly Perturbed Lattices" by Klatt [47], available here. See also how I use the function L in an application to generate locally random permutations, in Section 2.2. Now I can state two new theorems.

Theorem 4.6 *Let us assume that F_s has a derivative f_s (the density), continuous and strictly positive everywhere. For any $h \in \mathbb{Z}$, we have*

$$P(L(x)=k) = C \cdot f_s(x-t_k), \text{ with } C = \sum_{h\in\mathbb{Z}} f_s(x-t_h).$$

Proof
Let $B_\epsilon(x) = [x-\epsilon, x+\epsilon]$. We have

$$\lim_{\epsilon\to 0} \frac{P(X_k \in B_\epsilon(x))}{P(X_h \in B_\epsilon(x))} = \frac{f_s(x-t_k)}{f_s(x-t_h)}.$$

Thus $P(L(x)=k) \propto f_s(x-t_k)$, and the proportionality constant is such that the sum over all $k \in \mathbb{Z}$, must be one. ∎

Theorem 4.7 *The interarrival time T and K are connected by the following formula:*

$$P(T(\lambda,s) < y) = \sum_{k\neq 0} P(K=k) \int_{-\infty}^{\infty} F_s\Big(x+y-\frac{k}{\lambda}\Big) f_s(x)dx.$$

Proof
We have: $K=k$ if and only if k is the smallest index such that $X_k > X_0$. Thus,

$$\begin{aligned} P(T(\lambda,s)<y) &= P(X_K - X_0 < y) \\ &= \sum_{k\neq 0} P(K=k) P(X_k - X_0 < y) \\ &= \sum_{k\neq 0} P(K=k) \int_{-\infty}^{\infty} F_s\Big(x+y-\frac{k}{\lambda}\Big) f_s(x)dx. \end{aligned}$$

The last integral is the result of the convolution between the random variables X_k and $-X_0$. ∎

Theorem 4.7 provides us with a way to compute the $P(K=k)$'s. You need to solve a linear system with an infinite number of variables and an infinite number of equations. The unknowns are the $P(K=k)$'s. In practice, especially if $\lambda = 1$, you can just reduce it to $-n \leq k \leq n$, with $k \neq 0$ and $n = 10$, as $P(K=k)$ quickly decays to zero when k becomes large in absolute value. Pick up a different y in the integral, for each of the $2n$ equations, to get an invertible system.

The distribution of the interarrival times is combinatorial in nature, and in principle you could use the theory of order statistics [Wiki] to get the exact distribution. References on this topic includes [7, 17]. When the random variables are independently but not identically distributed, one may use the Bapat-Beg theorem [Wiki] to find the joint distribution of the order statistics of the sequence (X_k), and from there, obtain the theoretical distribution of T. This approach is difficult, and not recommended. Simulations are the preferred option.

4.8 Special Cases with Exact Formula

Again, let's focus on the counting random variable $N(B)$, with $B = [a,b]$ an interval with $a < b$. An exact, closed-form formula for the expectation and variance, is available in a few cases, in particular if F is a uniform or Laplace distribution. The formulas are complicated, and difficult to obain as they require laborious though trivial computations that could easily be automated with AI. Also, they are of not of much use since simulations are numerically stable and do well in this context, with few iterations, to estimate these quantities. So, I will only mention one of these exact formulas in Theorem 4.8. It is probably the most interesting one. It is in agreement with other results obtained previously.

Theorem 4.8 *If F_s is uniform on $[-s,s]$, $\lambda = 1$, and $B = [-s,s]$, then*

$$E[N(B)] = -1 + 2\sum_{0\leq k\leq 2s}\Big(1-\frac{k}{2s}\Big) = -1 + \frac{\lfloor 2s\rfloor(1+\lfloor 2s\rfloor)}{2s}.$$

The brackets represent the integer part function.

Proof
Let $\lambda = 1, B = [a, b]$ and $p_k = F_s(b-k) - F_s(a-k)$. Here

$$F_s(x-k) = \frac{1}{2} + \frac{x-k}{2s}, \text{ with } -s \leq x-k \leq s, \text{ and } s > 0.$$

We have two cases, each with three sub-cases:

If $b - a \leq 2s$ then
- If $a - s \leq k \leq b - s$ then $p_k = \frac{1}{2} - \frac{a-k}{2s}$.
- If $b - s \leq k \leq a + s$ then $p_k = \frac{b-a}{2s}$.
- If $a + s \leq k \leq b + s$ then $p_k = \frac{1}{2} + \frac{b-k}{2s}$.

If $b - a \geq 2s$ then
- If $a - s \leq k \leq a + s$ then $p_k = \frac{1}{2} - \frac{a-k}{2s}$.
- If $a + s \leq k \leq b - s$ then $p_k = 1$.
- If $b - s \leq k \leq b + s$ then $p_k = \frac{1}{2} + \frac{b-k}{2s}$.

If $k \notin [a-s, b+s]$, then $p_k = 0$. Let $B = [a, b]$. The above results can be used to compute (in closed form) the quantities

$$\text{E}[N(B)] = \sum_{k=-\infty}^{\infty} p_k, \quad \text{Var}[N(B)] = \sum_{k=-\infty}^{\infty} p_k(1-p_k).$$

In particular, if $a = -s$ and $b = s$, there are some simplifications, and we obtain the result announced in the theorem. ∎

Note that if $s/2$ is an integer, the above result is compatible with Theorem 38 since $\text{E}[N(B)] = 2s = b - a$. Also, as $s \to \infty$, $\text{E}[N(B)] \sim 2s = b - a$. In general though, $\text{E}[N(B)]$ is not an exact function of $\mu(B) = \lambda \cdot (b-a)$, confirming that the Poisson-binomial process is different from a Poisson process, and very much so in particular if F is the uniform distribution.

If F is the Laplace distribution, an exact, closed-form formula can also be obtained for $\text{E}[N(B)]$ and $\text{Var}[N(B)]$, and for higher moments. See Exercise 1 in Section 5.

4.9 Fundamental Theorem of Statistics

Many of the distributions used in this textbook for simulation purposes can easily be sampled using inverse transform sampling [Wiki]. The fact that – as in Table 3 – estimated quantities such as mean, variance or quantiles, converge to the desired theoretical values is due to the convergence of the empirical distribution [Wiki] (measured on the observations) to its theoretical limit (associated to the model). This can be re-stated as follows, and was used in particular to compute the moment generating function of the generalized logistic distribution in Section 2.1.1.

Theorem 4.9 *Let Z be a random variable with cumulative distribution function $F(z) = P(Z < z)$ with $z \in \mathbb{R}$, and quantile function $Q(u) = F^{-1}(u)$, with $0 \leq u \leq 1$. Let g be a measurable function. Then we have*

$$E[g(Z)] = \int_0^1 g(Q(u))du = \int_{-\infty}^{\infty} g(z)dF(z) = \int_{-\infty}^{\infty} g(z)f(z)dz.$$

Proof
Use the change of variable $u = F(z)$ in the leftmost integral. Then $Q(u)$ becomes $Q(F(z)) = F^{-1}(F(z)) = z$, du becomes $dF(z) = f(z)dz$, and the interval of integration changes from $[0, 1]$ to the entire real line. ∎

Here f is the density attached to F, assuming it exists. The righmost equality is well known, but the leftmost is not. Surprisingly, this unnamed, little known theorem, rarely if ever mentioned, has a crucial role. It is routinely and unconsciously used by all machine learning practioners almost on a daily basis, at least in the version that applies to empirical, observation-based statistics. The above version applies to theoretical (mathematical) statistics. I suggest to call it the quantile theorem.

For instance, the moment generating function of Z is defined as $\text{E}[\exp(tZ)]$. It can be computed via the quantile function Q, using $g(z) = \exp(tz)$, see Formula (16) for the generalized logistic distribution. See also Exercises 3 and 4 in Section 5, for a different application.

5 Exercises, with Solutions

While the purpose of these exercises is to strengthen the learning experience and to generate out-of-the-box thinking, perhaps even more importantly, they provide additional methodological and technical material, complementing and extending the main text.

Starred exercises are more difficult. Several of the problems require only simulations, statistical analysis, and testing hypotheses on a computer. They are marked as [S] and should help you hone your machine learning and computing skills; they may not be easier or less challenging than the mathematical problems. Exercises involving mathematics or probability theory are marked as [M], while those combining both simulations and mathematics are marked as [MS]. Solutions or hints are provided for each problem.

5.1 Full List

Table 7 provides a listing all the exercises. To access any exercise, click on its red number, to the left. The NN abbreviation stands for "nearest neighbors".

1	Point count, Laplace distribution	15	Distribution of NN distances
2 *	Convergence to Poisson process	16	Cell networks: coverage problem
3 *	Limit of generalized logistic distribution	17	Optimum circle covering of the plane
4	Small paradox	18	Interlaced lattices, lattice mixtures, NN
5	Exact distribution of interarrival times	19 *	Lattice topology and algebra
6 *	Retrieving F from interarrival times	20 **	NN graph: size of connected components
7 *	Poisson limit of Poisson-binomial distribution	21	NN graph: maximum clique problem
8	A few simple theorems	22	Computing moments using the CDF
9	Testing stationarity, independent increments	23	Simulations: generalized logistic distribution
10	Interdependencies in point counts	24	Riemann Hypothesis
11	Boundary effect	25 *	Convergence acceleration of math series
12	A curious, Poisson-like point process	26	Fast image filtering algorithm
13 *	Poisson-binomial process on the sphere	27 **	Confidence regions: theory, computations
14	Taxonomy of point processes	28 *	Minimum set covering 90% of a distribution

Table 7: List of exercises

5.2 Probability Distributions, Limits and Convergence

The focus here is on the distribution F including some of its limiting cases, the distribution of arrival times, and convergence to the Poisson process. The Laplace, generalized logistic, Borel, and Poisson-binomial distributions are investigated.

Exercise 1 [M] Point count, Laplace distribution. If F is a Laplace distribution and $\lambda = 1$, find $E[N(B)]$, where $B = [a, b]$ is an interval with $\lfloor a \rfloor \leq \lfloor b \rfloor < \lfloor a \rfloor + 1$. Here the brackets represent the integer part function, and $F_s(x) = F(x/s)$. See Theorem 4.8, solving the same problem with a uniform rather than Laplace distribution.

Solution

Let $p_k = F_s(b - k) - F_s(a - k)$ with $s > 0$, and let sgn stands for the sign function, with $\text{sgn}(0) = 0$. Here

$$F_s(x-k) = \frac{1}{2} + \frac{1}{2}\text{sgn}(x-k)\Big[1 - \exp\Big(-\frac{1}{s}\cdot|x-k|\Big)\Big]$$

We have three cases:

- If $k \leq a < b$ then $p_k = \frac{1}{2}\Big[\exp(-(a-k)/s) - \exp(-(b-k)/s)\Big]$
- If $a \leq k \leq b$ then $p_k = 1 - \frac{1}{2}\Big[\exp(-(b-k)/s) + \exp((a-k)/s)\Big]$
- If $a < b \leq k$ then $p_k = \frac{1}{2}\Big[\exp((b-k)/s) - \exp((a-k)/s)\Big]$

If $\lfloor a \rfloor \leq \lfloor b \rfloor < \lfloor a \rfloor + 1$, then the second case is empty and $\lfloor a \rfloor = \lfloor b \rfloor$. As a result, the computations simplify to

$$\begin{aligned}
2\mathrm{E}[N(B)] &= \alpha \sum_{k \leq a} \phi^k - \beta \sum_{k \leq a} \phi^k + \frac{1}{\beta} \sum_{k \geq b} \Big(\frac{1}{\phi}\Big)^k - \frac{1}{\alpha} \sum_{k \geq b} \Big(\frac{1}{\phi}\Big)^k \\
&= (\alpha - \beta)\Big[\sum_{k \leq a} \phi^k + \frac{1}{\alpha\beta} \sum_{k \geq b} \Big(\frac{1}{\phi}\Big)^k\Big] \\
&= (\alpha - \beta) \cdot \frac{\phi}{\phi - 1} \cdot \Big[\phi^{\lfloor a \rfloor} + \frac{1}{\alpha\beta}\Big(\frac{1}{\phi}\Big)^{\lfloor a \rfloor + 1}\Big]
\end{aligned}$$

where $\alpha = \exp(-a/s) \geq \beta = \exp(-b/s)$ and $\phi = \exp(1/s)$. We are dealing with geometric series, which are easily summable. The last equality is due to the fact that $\lfloor a \rfloor = \lfloor b \rfloor$. Note that b can not be an integer in this case, so a sum with integer index $k \geq b$ actually starts at $k = \lfloor b \rfloor + 1 = \lfloor a \rfloor + 1$. Also, when combining the various sums, make sure that their indices don't overlap, otherwise double counting will occur. This is not happening here.

Exercise 2 [M*] Convergence to Poisson process. Prove Theorem 4.5 in Section 4.

Hint

It is about the same level of difficulty as proving the Central Limit Theorem [Wiki] Central Limit Theorem (CLT). If understanding a proof of the CLT is beyond your mathematical level, you will find this exercise really difficult. However, you can focus on just proving that the interarrival time $T(\lambda, s)$ follows an exponential distribution, which in turn characterizes the Poisson process. This is not easy either. If you look at my proof, you will notice that a some point, I approximate a sum $\sum_{-n}^{n}$ by an integral $\int_{-n}^{n}$ as $n \to \infty$, implicitly using a version of the Euler-Maclaurin summation formula [Wiki] in its simplest form.

Exercise 3 [M*] Limit of generalized logistic distribution. Compute the expectation of the generalized logistic distribution, when $\alpha = 1$ and $1/\beta$ is a positive integer. If $\alpha = 1$ and $\tau = e^{1/\beta}$, prove that $\frac{1}{\beta\rho}(\mathrm{E}[Z] - \mu) \to -\pi^2/6$ as $\beta \to 0$. See Formula (13) for the cumulative distribution function.

Solution Instead of using Formula (14) to compute the expectation, I use Formula (15), with $r = 1$. Using Formula (12), the expectation can be rewritten as

$$\mathrm{E}[Z] = \mu + \rho \int_0^1 \log\Big(\frac{\tau u^m}{1 - u^m}\Big) du = \mu - \rho\Big[m \log \tau + \int_0^1 \log(1 - u^m) du\Big],$$

where $m = 1/\beta$ is an integer. Also, $1 - u^m$ is a polynomial of degree m, and its roots are the m-th roots of 1 in the complex plane (see here), that is

$$(1 - u^m) = -\prod_{k=0}^{m-1}\Big[u - \exp\Big(\frac{2k\pi i}{m}\Big)\Big].$$

Thus,

$$\log(1 - u^m) = \log(-1) + \sum_{k=0}^{m-1} \log\Big[u - \exp\Big(\frac{2k\pi i}{m}\Big)\Big].$$

Since $\log(-1) = \pi i$ and $\int \log(u - c) du = (u - c)\log(u - c) - u$ if c is a constant (whether complex or real), we finally have

$$\mathrm{E}[Z] = \mu - \rho m \log \tau - \rho \pi i - \rho \sum_{k=0}^{m-1}\Big[(1 - c_{k,m})\log(1 - c_{k,m}) - 1 + c_{k,m} \log(-c_{k,m})\Big]$$

where $c_{k,m} = \exp(2k\pi i/m)$. This involves computing complex logarithms [Wiki]. When combining the real and imaginary parts from all the terms, only real numbers are left. This tedious computation is best achieved using some automated tool. See the result for $\mu = 0, \tau = 1, \rho = 1$ and $m = 8$, using the online version of Mathematica, here. In this case, the final result is

$$-4 \log 2 - (\pi/2)\cot(\pi/8) - \sqrt{2}\log(\cot(\pi/8)).$$

Assuming $\alpha = 1$ and $\beta = 1/m$, when $m \to \infty$, we have the asymptotic expansion

$$\mathrm{E}[Z] = \mu + \rho\Big[\log \tau - m - \frac{\xi}{m} + o\Big(\frac{1}{m}\Big)\Big], \text{ with } \xi = \lim_{m \to \infty}\Big(m \int_0^1 \log(1 - u^m) du\Big) = -\frac{\pi^2}{6}. \tag{41}$$

The value of ξ was obtained by replacing $\log(1-u^m)$ by its Taylor series in the integral, then integrating term by term, and finally taking the limit as $m \to \infty$. It can also be obtained with the change of variable $v = u^m$ in the integral. Let $\alpha = 1$ and $\tau = e^{1/\beta} = e^m$. From (41), we get: $\text{E}[Z] \to \mu$ and $\frac{1}{\beta\rho}(\text{E}[Z]-\mu) \to \pi^2/6$ as $\beta \to 0$.

Exercise 4 [MS] Small paradox. Let Z_β be a random variable with generalized logistic distribution, with $\mu = 0$, $\rho = \alpha = 1$ and $\tau = e^{1/\beta}$. Using simulations based on Formula (12) for the quantile function $Q(u)$, with u a uniform deviate on $[0,1]$, try to guess the expectation and variance of the limit random variable $Z_* = \lim_{\beta\to 0} \beta^{-1} Z_\beta$. The exact values are respectively zero and one. Using numerical approximations, show that $\lim_{\beta\to 0} \beta^{-1}\text{E}[Z_\beta] \approx 1.645$. All of this can be done in Excel using the `rand` function. Then obtain the exact values for the three quantities in question. The last one, equal to $\pi^2/6$, was computed in Exercise 3. Thus, in this case, the expectation and limit operators can not be swapped (one yields the answer 0, the other one yields $\frac{\pi^2}{6}$). This is because the limiting distribution $P(Z_* < z)$ is truncated, as shown in the solution below.

Solution Despite the appearance, this is an easy exercise. As in Exercise 3, let $m = 1/\beta$ and $m \to \infty$. The problem here, when approximating

$$\beta^{-1}\text{E}[Z_\beta] = m\int_0^1 Q(u)du = m\int_0^1 \Big[\log(\tau u^m) - \log(1-u^m)\Big]du = -m\int_0^1 \log(1-u^m)du,$$

is that the computation of $\log(1-u^m)$ is numerically unstable [42] when $0 < u < 1$ and m is large, resulting in erroneous results, whether you do it in Excel or Python. The problem is said to be ill-conditioned. To avoid this problem, use the change of variable $v = u^m$, yielding

$$-m\int_0^1 \log(1-u^m)du = -\int_0^1 \frac{\log(1-v)}{v^{1-1/m}}dv \to -\int_0^1 v^{-1}\log(1-v)dv = \frac{\pi^2}{6} \text{ as } m \to \infty.$$

Base your Excel computations on the last integral, using a sum to approximate it. Now it works!

The fact that $\text{E}[Z_*] = 0$ and $\text{Var}[Z_*] = 1$ will be apparent from your simulations involving the quantile function $Q(u)$. However, to prove it rigorously, rather than using Q, it is easier to work with the CDF $P(Z_\beta < z)$, with $\mu = 0, \rho = \alpha = 1, \beta = 1/m$ and $\tau = e^m$, using Formula (13):

$$P(Z_* < z) = \lim_{m\to\infty} P(Z_{1/m} < mz) = \lim_{m\to\infty} \frac{1}{(1+\tau^m \exp(-mz))^m} = \lim_{m\to\infty} \frac{1}{(1+\exp[m(1-z)])^m}.$$

If $z > 1$, the above limit is one, otherwise it is equal to $\exp[-(1-z)]$. The density f_{Z_*} (the derivative of the CDF) is also equal to $f_{Z_*}(z) = \exp[-(1-z)]$ with support domain $z < 1$. Thus we have $\text{E}[Z_*] = \int_{-\infty}^1 z f_{Z_*}(z)dz = 0$ and $\text{Var}[Z_*] = \text{E}[Z_*^2] = \int_{-\infty}^1 z^2 f_{Z_*}(z)dz = 1$.

Exercise 5 [M] Exact distribution of interarrival times. Find the distribution of the interarrival times $T(\lambda, s)$ if all the points are ordered, that is, if $X_k < X_{k+1}$ for all $k \in \mathbb{Z}$. This happens when s is small enough, and the tail of F is not too thick.

Solution

We have $X_k = \frac{k}{\lambda} + sZ_k$ and $X_{k+1} = \frac{k+1}{\lambda} + sZ_{k+1}$ where Z_k, Z_{k+1} are two independent random variables of distribution F. Thus the interarrival time $X_{k+1} - X_k$ has the same distribution as $T = \frac{1}{\lambda} + s(Z_{k+1} - Z_k)$ and does not depend on k. You may as well use $k = 0$ for its computation. The result is

$$P(T < y) = P\Big(\frac{1}{\lambda} + s(Z_1 - Z_0) < y\Big) = P\Big(Z_1 - Z_0 < \frac{y - 1/\lambda}{s}\Big) = \int_{-\infty}^{\infty} F\Big(x + \frac{y-1/\lambda}{s}\Big)f(x)dx$$

where f is the density attached to F. Since f is symmetric and centered at the origin, the distribution $P(T < y)$ is the self-convolution of F [Wiki], also denoted as $F * F$.

Examples:

- If F is normal with zero mean and unit variance, then T is almost normal, with mean $1/\lambda$ and variance $2s$, assuming s is small compared to $1/\lambda$. But T's distribution can not be exactly normal, because in this case, the X_k's can not all be *perfectly* naturally ordered unless $s = 0$ (then $X_k = k/\lambda$).
- If F is uniform on $[-1, 1]$ then T has a symmetric triangular distribution [Wiki] of mean $1/\lambda$ and support domain $[\frac{1}{\lambda} - 2s, \frac{1}{\lambda} + 2s]$. This is the exact solution if $0 \le s < \frac{1}{2\lambda}$. In this case, the X_k's are all naturally ordered.

For a more formal result, see Theorem 4.4. See also Exercise 6.

Exercise 6 [M*] Retrieving F from the interarrival times distribution. I assume here that F has a density f. Given the limit distribution of the standardized interarrival times, the purpose is to retrieve the distribution of F. If you are familiar with the concept of characteristic function [Wiki], this exercise is easy. If not, you should first get familiar with this concept. Thus this exercise is marked as difficult.

The standardized interarrival times is defined as $\frac{1}{s}[T(\lambda,s)-\frac{1}{\lambda}]$ and has zero expectation by virtue of Theorem 4.3. By virtue of Theorem 4.2, it can be rewritten as $\frac{1}{\lambda s}[T(1,\lambda s)-1]$. Its limit, as $s\to 0$, is denoted as T^*. One of the simplest cases, besides Gaussian and Cauchy, is the following: If T^* has a standard Laplace distribution [Wiki] (that is, symmetric centered at zero and with variance $\frac{\pi^2}{3}$), show that F is a modified Bessel distribution of the second kind (see reference [63], available online here). Note that as a consequence of L'Hôpital's rule [Wiki], T^* is the derivative of $T(\lambda,s)$ with respect to s, evaluated at $s=0$.

Solution

By virtue of Theorem 4.4, we have

$$P(T^*<y)=\int_{-\infty}^{\infty}F(y-x)f(x)dx,$$

which is a convolution of F with itself. Thus T^* has the distribution of the sum of two independent random variables, say Z_1,Z_2, of distribution F. Its characteristic function is therefore

$$\text{E}[\exp(-itT^*)]=\frac{1}{1+t^2}=\text{E}[\exp(-itZ_1)]\times\text{E}[\exp(-itZ_2)]=\Big(\text{E}[\exp(-itZ_1)]\Big)^2.$$

Thus $\text{E}[\exp(-itZ_1)]=(1+t^2)^{-1/2}$. Taking the inverse Fourier transform to retrieve the density of Z_1, which is the density attached to F, one finds

$$f(x)=\frac{1}{2\pi}\int_{-\infty}^{\infty}\frac{\cos(tx)}{\sqrt{1+t^2}}dt=\frac{1}{\pi}K_0(x),$$

where K_0 is the modified Bessel function of the second kind [Wiki]. More about the Laplace distribution and its generalization can be found in [49]. The cases when T^* is Gaussian or Cauchy are easy because these distributions belong to stable families of distributions [Wiki]: in that case, F is respectively Gaussian or Cauchy.

Exercise 7 [M*] Poisson limit of Poisson-binomial distribution. Theorem 2.1 shows that a particular case of the Poisson-binomial distribution converges to the Poisson distribution. In the proof, I established the values of $P(N=0),P(N=1)$ for the counting random variable N. I also stated (without proving it), that as $n\to\infty$ and $m/n\to\alpha$, we have $P(N=k)\to q_0\mu^k/k!$ for all positive integers k. The purpose of this exercise is to prove this latter statement. This in turn completes the proof of Theorem 2.1. The notations refer to the theorem in question. In particular, $q_0=P(N=0)$ and $\mu=P(N=1)/P(N=0)$. This exercise reveals the true combinatorial nature of the Poisson-binomial distribution, in all its complexity. This is also related to Le Cam's inequality.

Solution

For $P(N=0)$ and $P(N=1)$, see proof of Theorem 2.1. Let $p_k=1/(n+k)$ as in the proof of Theorem 2.1. We have, with the convention that a sum or product such as $\sum_{k_2\neq k_1}$ is over $k_2=1,\dots,k_1-1,k_1+1,\dots,m$:

$$\begin{aligned}P(N=2)&=\sum_{k_1=1}^{m}p_{k_1}\sum_{k_2\neq k_1}p_{k_2}\prod_{k_1\neq k\neq k_2}(1-p_k)\\&=\sum_{k_1=1}^{m}\sum_{k_2\neq k_1}\frac{p_{k_1}}{1-p_{k_1}}\frac{p_{k_2}}{1-p_{k_2}}\prod_{k=1}^{m}(1-p_k)\\&=q_0\sum_{k_1=1}^{m}\sum_{k_2\neq k_1}\frac{p_{k_1}}{1-p_{k_1}}\frac{p_{k_2}}{1-p_{k_2}}.\end{aligned}\tag{42}$$

Let S denotes the double sum in Formula (42). We have

$$\begin{aligned}S&=\frac{1}{2}\sum_{k_1=1}^{m}\Big[\Big(\sum_{k_2=1}^{m}\frac{p_{k_1}}{1-p_{k_1}}\frac{p_{k_2}}{1-p_{k_2}}\Big)-\Big(\frac{p_{k_1}}{1-p_{k_1}}\Big)^2\Big]\\&=\frac{1}{2}\Big[\sum_{k_1=1}^{m}\sum_{k_2=1}^{m}\frac{p_{k_1}}{1-p_{k_1}}\frac{p_{k_2}}{1-p_{k_2}}\Big]-\frac{1}{2}\sum_{k_1=1}^{m}\Big(\frac{p_{k_1}}{1-p_{k_1}}\Big)^2\\&=\frac{1}{2}\Big(\sum_{k=1}^{m}\frac{p_k}{1-p_k}\Big)^2-\frac{1}{2}\sum_{k=1}^{m}\Big(\frac{p_k}{1-p_k}\Big)^2=\frac{\mu^2}{2}-\frac{1}{2}\sum_{k=1}^{m}\Big(\frac{p_k}{1-p_k}\Big)^2\end{aligned}\tag{43}$$

with

$$\mu = \sum_{k=1}^{m} \frac{p_k}{1-p_k} = \sum_{k=1}^{m} \frac{1}{n+k-1} = \sum_{k=n}^{m-1} \frac{1}{k} \to \log\alpha \text{ as } n\to\infty, \frac{m}{n} \to \alpha > 1,$$

and

$$\sum_{k=1}^{m} \Big(\frac{p_k}{1-p_k}\Big)^2 = \sum_{k=1}^{m} \frac{1}{(n+k-1)^2} \to 0 \text{ as } n\to\infty.$$

Note that the fraction $\frac{1}{2}$ in Formula (43) is required to eliminate double counting the products in the double summation. For $P(N=3)$, we have a triple summation over indices k_1, k_2, k_3, and because there are $3! = 6$ ways to re-arrange distinct k_1, k_2, k_3, the fraction $\frac{1}{2} = \frac{1}{2!}$ becomes $\frac{1}{3!}$; likewise, because of the triple product, μ^2 becomes μ^3.

Now I proved that $P(N=2) = q_0\frac{\mu^2}{2!}$, and provided hints as to why $P(N=3) = q_0\frac{\mu^3}{3!}$. The general case if left to the reader.

5.3 Features of Poisson-binomial Processes

In this section, I discuss properties of point processes, such as ergodicity, homogeneity, stationarity and independent increments. One exercise deals with boundary effects, a result of data censoring. A curious Poisson-like point process is also investigated.

Exercise 8 [M] A few simple theorems. Prove all theorems in Section 4, except Theorem 4.5.

Hint

Of course you can just look at the proof that I provided for each theorem. However, it is a much better learning experience to try to prove them on your own without reading my solution, and possibly to generalize the theorems, for instance from one dimension to any dimension, if applicable. Your proofs might even be shorter, more rigorous, complete or elegant than mines. In fact, mines are mostly sketch proofs. For each of the theorems in question, some key trick is required to make progress towards a short, easy but subtle proof.

Exercise 9 [S] Ergodicity, independent increments. Test on simulated data (for various realizations of Poisson-binomial processes) if and when the following assumptions are violated, depending on s, F and other parameters.

- Independent increments: point counts in non-overlapping intervals are independent.
- In two dimensions, zero correlation between the X and Y coordinates of a point.
- Ergodicity for interarrival times, in one dimension.
- Homogeneity: the point density is *statistically* the same anywhere on the plane or on the real line.
- Anisotropy: the point density, even if non homogeneous, does not show directional trends.
- Stationarity: the point count in $[a, b]$ is *statistically* the same as in $[a + c, b + c]$, regardless of a, b, c.
- Aperiodicity: the point density on the real line does not exhibit periodic behavior.

To perform the simulations to test the various assumptions, you can use the source code in Section 6.2.

Solution

Some of these assumptions (aperiodicity, stationarity) are violated when the scaling factor s is close enough to zero. If s is large (say $s = 40, \lambda = 1$), the process is not statistically different from a stationary Poisson point process, so no statistical test will be able to detect violations, even if present. That said, interarrival times exhibit ergodicity and independence. For instance, while T is defined as the distance between X_0 and its closest neighbor to the right, you can replace X_0 by X_1, X_2, or any X_k, and T's distribution remains unchanged. There is also stationarity in the following sense, regardless of s: point counts in $[a, b]$ and $[a + c, b + c]$ have the same distribution if c is a multiple of $1/\lambda$.

The largest departure from stationarity occurs with small s, and using a uniform distribution for F. If F is Cauchy, things look prettier (more stationary) as F has a thick tail and does a better job at mixing the points. To illustrate the non-stationarity, use $\lambda = 1.4, s = 0.15$ with the logistic distribution for F. Let $B_1 = [0, 0.8]$ and $B_2 = [6, 6.8]$. Then $\text{Var}[N(B_1)] \approx 0.506 \neq \text{Var}[N(B_2)] \approx 0.312$. Now if you increase the scaling factor from $s = 0.15$ to $s = 0.6$, the process is almost stationary. In particular the two variances in question range from 0.878181 to 0.878193 depending on the interval. The same is true for other statistics. For all practical purposes, the distribution of $N([a, b])$ depends only on $b - a$ if $s \geq 0.6$. Statistical tests would not be able to detect the minuscule lack of stationarity.

I used the exact Formula (5) to compute point count variances. This formula is implemented in the source code in Section 6.2.1. In Exercise 10, I prove non-independence for point counts over non-overlapping domains.

Exercise 10 [M] Joint distribution of point counts. Let B_1, B_2 be non-overlapping domains. For Poisson point processes, the point counts $N(B_1)$ and and $N(B_2)$ are independent random variables. Is this also true for Poisson-binomial processes? Consider the one-dimensional case, with $B_1 = [a, b[$, $B_2 = [b, c[$.

Solution

The answer is negative, although the dependence is weak [Wiki]. Let $B_{12} = B_1 \cup B_2$, $q_1 = P[N(B_1) = 0]$, $q_2 = P[N(B_2) = 0]$ and $q_{12} = P[N(B_{12}) = 0] = P[N(B_1) = 0, N(B_2) = 0]$. It suffices to prove that in general, $q_{12} \neq q_1 q_2$.

Let X_k, $k \in \mathbb{Z}$ be the points of the Poisson-binomial process. According to Formula (6), we have:

$$q_{12} = \prod_{k=-\infty}^{\infty} \Big[1 - F\Big(\frac{c - k/\lambda}{s}\Big) + F\Big(\frac{a - k/\lambda}{s}\Big)\Big]$$

$$q_1 = \prod_{k=-\infty}^{\infty} \Big[1 - F\Big(\frac{b - k/\lambda}{s}\Big) + F\Big(\frac{a - k/\lambda}{s}\Big)\Big]$$

$$q_2 = \prod_{k=-\infty}^{\infty} \Big[1 - F\Big(\frac{c - k/\lambda}{s}\Big) + F\Big(\frac{b - k/\lambda}{s}\Big)\Big]$$

There is no reason why we would have $q_{12} = q_1 q_2$. For instance, if F is the logistic distribution, $\lambda = 1.4, s = 0.29$, $a = 0, b = 0.8$ and $c = 1.6$, we have (approximately) $q_1 = 0.2329$, $q_2 = 0.2306$, $q_{12} = 0.0177$, and $q_1 q_2 = 0.0537$.

However the dependence is weak. Also, we have asymptotic independence, with full independence when $s = \infty$, thanks to Theorem 4.5. Formula (6) is implemented in the source code, in Section 6.2.1. See also Section 3.1.3, featuring a test of independence for the point counts.

Exercise 11 [S] Boundary effect. The purpose is to assess the impact of the boundary effect, in one dimension. Assuming $\lambda = 1$, use a small value of n, say $n = 300$, to generate $2n + 1$ points X_k, $k = -n, \dots, n$ of a Poisson-binomial process. Estimate $\mathrm{E}[T]$, the expectation of the interarrival times, using all the $2n + 1$ points. Do the same, this time using $N = 10^4$, to generate $2N + 1$ points X_k, $k = -N, \dots, N$ of the same Poisson-binomial process. But only use the $2n + 1$ innermost points (closest to zero) still with $n = 300$, in your estimation of $E[T]$. These $2n + 1$ points won't be the same as in the first simulation. Also, some closest neighbors won't be among the $2n + 1$ innermost points but instead, in the larger set of $2N + 1$ points. Now your estimate takes into account nearest neighbors that were unobserved in the first simulation (called censored data) because they were outside the boundary. Compare your two estimates of $\mathrm{E}[T]$. The first one is slightly biased due to boundary effects, the latter one almost has no bias. Compare the impact of using a Cauchy versus a uniform distribution for F, by looking at the loss of accuracy when estimating $\mathrm{E}[T]$ based on a single realization of the process.

Hint

Try a simulation with $s = 0.5$, and one with $s = 10$. A large s, a thick tail (Cauchy versus uniform F), or a small value of n, all magnify the boundary effect, resulting in loss of accuracy in the estimates. Source code to compute $\mathrm{E}[T]$ can be found in Section 6.2.2.

Exercise 12 [M] A curious, Poisson-like point process. If we use a uniform distribution for F, and $s = \frac{1}{2\lambda}$, is the resulting process a stationary Poisson process? What if we use a mixture of m such processes in equal proportions, called a m-mixture? (see Exercises 18 and 19). Assume here that we work with 2-dimensional Poisson-binomial point processes.

Solution

In this case, each point (X_h, Y_k) of the process is uniformly distributed on a square with sides of length $1/\lambda$ and centered at $(h/\lambda, k/\lambda)$. The support domains of these uniform distributions form a partition [Wiki] of $\mathbb{R}^2$: they don't overlap, and there is also no empty space left. So the points of the process are uniformly and independently distributed on each square B of area $1/\lambda^2$. But there is only one point in any such B. The process is a Poisson-binomial point process of intensity λ (by construction), but it can not be a standard Poisson process. If we mix m such processes, the resulting process has a point count $N(B)$ with identical binomial distributions [Wiki] on any square B of area $1/\lambda^2$, with $N(B) \in \{0, \dots, m\}$ and $\mathrm{E}[N(B)] = 1$. It is is not a Poisson process either since $N(B)$ does not have a Poisson distribution, though it is getting close.

Exercise 13 [S*] Poisson-binomial process on the sphere. Build a Poisson-binomial process on the sphere. You can start with a circle first, a cube or a torus. Study its properties, such as the distribution of nearest neighbor distances or the size of connected components, via simulation. Note that in this case, the point process has a finite number of points. See also "Nearest Neighbor and Contact Distance Distribution for Binomial Point Process on Spherical Surfaces" [75], avaiable online here.

Solution

The first step is to define a lattice on the sphere. One way to do it is to build an inscribed polyhedron inside the sphere [Wiki], and use its vertices as the lattice locations in the lattice space. See [65], available online here. An easier way is as follows:

- plot longitudes and latitudes at equally spaced angles,
- the points where they intersect are the lattice locations,
- the angle between two successive parallels (latitude or longitudes) is the intensity.

The disadvantage of this method is that it creates two poles, and the lattice locations are not evenly distributed on the sphere. The resulting process is not homogeneous. For a solution with evenly distributed lattice locations, see here.

Now around each lattice location, generate a random point on the surface of the sphere. The point is specified by two independent random variables: an angle θ uniformly distributed on $[0, 2\pi]$, and a radius R measuring the distance to the lattice location on the surface of the sphere. It makes sense to require $R \leq \pi\rho$, where ρ is the radius of the sphere. The scaling factor can be defined as $s = \mathrm{E}[R]$. Note that there are no boundary effects here. The next step is the create clusters on the sphere. See [33], available online here. Also, one can study the conditions to obtain convergence to a stationary Poisson point process on the sphere.

Another possible generalization is random lines. In two dimensions, a line is characterized by two quantities: its distance R to the origin, and its orientation θ. A similar methodology can be used to produce a Poisson-binomial line process, with the angle θ uniformly distributed on $[0, 2\pi]$. In this case, the lattice space could be $(\mathbb{Z}/\lambda) \times (\mathbb{Z}/\lambda)$, where λ is the intensity. Also see "Generating stratified random lines in a square" [70], available online here. This is a typical stochastic geometry problem.

Exercise 14 [S] Taxonomy of point processes. The purpose of this exercise is to prove that each type of point process studied in details in this textbook, is unique. In other words, the overlap between the different classes of point processes is small, despite model identifiability issues. Here, I ask you to verify, via examples, that m-interlacings defined in Section 1.5.3 are different from m-mixtures, stationary Poisson processes, Poisson-binomial point processes, and the radial cluster processes discussed in Section 2.1.

Solution

As usual, the differences are most striking when the scaling factor s is very small. In that case, for m-interlacings, each lattice location in the lattice space has exactly m points of the process clustered around it. For Poisson-binomial and m-mixtures, that number is one. For radial cluster processes (with a Poisson-binomial parent process), the number in question is random and depends on the location. For Poisson point processes (the limit of some of these processes when $s \to \infty$) the underlying lattice space becomes meaningless.

5.4 Lattice Networks, Covering Problems, and Nearest Neighbors

In this section dealing with two-dimensional problems, various lattices underlying Poisson-binomial processes are explored, including lattice properties and the construction of hexagonal lattices using superimposed rectangular lattices. Another problem with applications to cellular networks is covering of the plane with circles, when the centers are distributed as a Poisson process, or on a fixed lattice. Finally, nearest neighbor distances and connected components are analyzed, giving this section a "graph theory" flavor.

Exercise 15 [MS] Distribution of nearest neighbor distances. In two dimensions, $T(\lambda, s)$ represents the distance between a point of the process and its nearest neighbor.

- Prove that when $s \to \infty$, the limiting distribution of T is Rayleigh [Wiki] of mean $\frac{1}{2\lambda}$.
- Show by simulations or logical arguments, that unlike in the one dimensional case (see Theorem 4.3), T's expectation depends on s.
- Also, show that depending on F, the maximum nearest neighbor distance, computed over the infinitely many points of the process, can have a finite expectation. Is this true too when $s \to \infty$, that is, for stationary Poisson point processes?
- Finally, what is T's distribution if T is replaced by the distance between an arbitrary location in $\mathbb{R}^2$, and its closest neighbor among the points of the process?

Solution

In two dimensions, the fact that $\mathrm{E}[T(\lambda, s)]$ depends on s, is obvious: if $s = 0$, it is equal to $\frac{1}{\lambda}$, and if $s = \infty$, it is equal to $\frac{1}{2\lambda}$. Between these two extremes, there is a continuum of values, of course depending on s. The maximum nearest neighbor distance (over all the infinitely many points) always has a finite expectation if F is uniform, regardless of $s < \infty$. To the contrary, for a Poisson point process, the maximum is infinite, see here.

Now let's prove that T has a Rayleigh distribution when $s = \infty$, corresponding to a Poisson process of intensity λ^2. We have $P(T > y) = P[N(B) = 0]$, where B is a disc of radius y centered at an arbitrary point of the process, and N is the point count, with an exponential distribution of mean $\lambda^2\mu(B)$ with $\mu(B) = \pi y^2$ being the area of B. Thus $P(T > y) = \exp(-\lambda^2\pi y^2)$, that is, $P(T < y) = 1 - \exp(-\lambda^2\pi y^2)$. This is the CDF of a Rayleigh distribution of mean $\frac{1}{2\lambda}$.

Exercise 16 [M] Cell networks: coverage problem. Points are randomly distributed on the plane, with an average of λ points per unit area. A circle of radius R is drawn around each point. What is the proportion of the plane covered by these (possibly overlapping) circles? What if R is a random variable, so that we are dealing with random circles? Such stochastic covering problems are part of stochastic geometry [Wiki] [22, 74]. See also Hall's book on coverings [39]. Applications include wireless networks [Wiki].

Solution

The points are distributed according to a Poisson point process of intensity λ. The probability that an arbitrary location x in the plane is not covered by any circle, is the probability that there is zero point from the process, in a circle of radius R centered at x. This is equal to $\exp(-\lambda\pi R^2)$. Thus the proportion of the plane covered by the circles is $1 - \exp(-\lambda\pi R^2)$. Now, let's say that we have two types of circles: one with radius R_1, and one with radius R_2, each type equally likely to be picked up. This is like having two independent, superimposed Poisson processes (see Section 1.5.3), each with intensity $\lambda/2$, one for each type of circle. Now the probability p that x is not covered by any circle is thus a product of two probabilities:

$$p = \exp\Big(-\frac{\lambda}{2}\pi R_1^2\Big) \times \exp\Big(-\frac{\lambda}{2}\pi R_1^2\Big) = \exp\Big(-\lambda\pi\frac{R_1^2 + R_2^2}{2}\Big).$$

You can generalize to m types of circles, each type with a radius R_k and probability p_k to be picked up, with $1 \leq k \leq m$. It leads to

$$1 - p = 1 - \exp\Big[-\lambda\pi\sum_{k=1}^{m} p_k R_k^2\Big], \tag{44}$$

which is the proportion of the plane covered by at least one circle. If R, the radius of the circle, is a continuous random variable, the sum in Formula (44) must be replaced by $\mathrm{E}[R^2]$. A related topic is the smallest circle problem [Wiki].

Exercise 17 [M] Optimum circle covering of the plane. This is an old problem, mentioned by Kershner in 1939 [46], revisited in 1971 by Williams [78], and still active today, see [32] (available online here) and [67] (available online here). Unlike in Exercise 16, the slightly overlapping circles of fixed radius, covering the entire plane, have centers located on a lattice rather than being the points of a Poisson process; in other words, the scaling factor s of the underlying Poisson-binomial process is zero (the point process reduces to its lattice space).

Applications include cellular network coverage, optimum location of sensor devices, and supply chain optimization such as optimum packing [Wiki]. The circle covering problem [Wiki] consists of finding the lattice that achieves optimum coverage: each location in the plane is covered by an average of $p > 1$ circles; the optimum is reached when p is minimum. Compute p both for the hexagonal lattice, and for the square lattice. Note that throughout this textbook, I worked with Poisson-binomial processes defined on a square lattice, except when considering lattice rotations, stretching, and superimposition in Section 1.5.3.

Solution

Let's start with circle centers located on a square lattice. For full coverage of the plane with as little overlapping as possible, the circles must be the smallest ones covering a square: the four vertices of the square must lie on the circle boundary, and the centers (both for the circle and square) coincide. For a unit square, such a circle must have a radius equal to $\sqrt{2}/2$ and an area equal to $\pi/2$. It is easy to see that $p = \pi/2 \approx 1.571$. This is illustrated here. For an hexagonal lattice [Wiki], the circle must be the smallest one covering an hexagon and having the same center as the inscribed hexagon [Wiki]. Computations (see [46]) show that $p = 2\pi/\sqrt{27}$. This is indeed the minimum possible value for p. There are only five types of regular lattices, called Bravais lattices [Wiki]. The hexagon is the regular polygon with the maximum number of sides, among those able to produce a *regular* Voronoi tessellation [Wiki], and thus results in the optimum lattice and minimum p.

Exercise 18 [S] Interlaced lattices, lattice mixtures and nearest neighbors. This is an additive number theory problem [Wiki], see also [64]. Let us consider a mixture (called m-mixture) or superimposition (called m-interlacing) of m shifted two-dimensional Poisson-binomial processes $M_1, \dots, M_m$ with scaling factor $s = 0$. Thus, these are non-random processes, where the state space of the i-th process M_i corresponds to its shifted lattice space: $(X_{ih}, X_{ik}) = (\mu_i + h/\lambda, \mu'_i + k/\lambda)$ for each point (X_{ih}, X_{ik}) of M_i, with $(h, k) \in \mathbb{Z}^2$ and (μ_i, μ'_i) is the shift parameter vector of M_i, depending only on i. Assume that each M_i has intensity $\lambda = 1$. Perform simulations to compare the distribution of nearest neighbor distances between m-interlacings and m-mixtures. More specifically, we are interested in the number of unique values that it can take. Conclude from this experiment that m-interlacings with small s, are less "random" than m-mixtures with the same s. Mixtures and superimposition of shifted processes are discussed in Section 1.5.3 and 1.5.4. By nearest neighbors, I mean among points of the m-mixture or m-interlacing, not between an arbitrary location and a point of the process, nor within each individual M_i taken separately.

Solution

For m-interlacings with $s = 0$, we have exactly m points $P_1, \dots, P_m$ in the square $[0, \frac{1}{\lambda}[\times[0, \frac{1}{\lambda}[$ (or in any square of same area, for that matter), and thus m pairs $\{P_i, P'_i\}$ $(i = 1, \dots, m)$ where P'_i is the nearest neighbor (NN) to P_i. Thus we have at most m distinct NN distances $||P_i - P'_i||$. So for m-interlacings with $s = 0$, the maximum number of unique values for the NN distance is m.

For m-mixtures, the situation is different. Now we have between 1 and m points in the square $B = [0, \frac{1}{\lambda}[\times[0, \frac{1}{\lambda}[$, assuming $s = 0$. Each of these points has one NN, possibly in the same square or in an adjacent square. For instance, if $P_i \in B$, it has one NN: a point $P_j \in B$ or a shifted version of P_j in an adjacent square. All combinations $i, j \in \{1, \dots, m\}$ are possible, and will necessarily show up (with probability one) in some squares of same area $1/\lambda^2$. Thus the number of unique NN distances is at least $m^2 - 1$, and at most $m \cdot (4m - 1)$. The "minus one" is because a point can not be its NN, that is, $P_i \neq P'_i$.

Simulations confirm these findings, both for m-interlacings and m-mixtures. It is assumed here that the shift vectors (μ_i, μ'_i) are arbitrary, as if they were randomly generated.

Exercise 19 [SM*] Lattice topology and algebra Using a superimposition of m stretched shifted Poisson-binomial processes $M_1, \dots, M_m$, denoted as M and called an m-interlacing in Exercise 18, build a point process that has a regular hexagonal lattice as its lattice space, with m as small as possible. Note that each M_i has a rectangular lattice space. Superimposed stretched shifted processes are defined in Section 1.5.3. When $s = 0$, M is identical to its fixed (non-random) hexagonal lattice space, see left plot in Figure 2. It is also clear from Figure 2 that each point of M has exactly 3 nearest neighbors. To the contrary, in a square lattice, each point (called vertex in graph theory) has 4 nearest neighbors. In a rectangular (non-square) lattice, each vertex has 2 nearest neighbors. Is it possible to build a lattice where each vertex has 5 or 6 nearest neighbors? A line joining two nearest neighbor vertices is called an edge. In Figure 2, all edges have the same unit length. Use Formulas (8) and (9) to generate a realization of M. The challenge is to find the minimum m and then identify the parameters λ, λ' and μ_i, μ'_i $(i = 1, \dots, m)$ resulting in a regular hexagonal lattice when $s = 0$. By regular, I mean that all edges have the same length, and only one regular polygon is used in the construction (in our case, an hexagon).

Solution

The solution can be found in Section 1.5.3. I used $m = 4$, and I don't think you can use a smaller m. The parameters are $\lambda = 1/3$, $\lambda' = \sqrt{3}/3$, $\mu_1 = 0, \mu_2 = 1/2, \mu_3 = 2, \mu_4 = 3/2$ and $\mu'_1 = 0, \mu'_2 = \sqrt{3}/2, \mu'_3 = 0, \mu'_4 = \sqrt{3}/2$. You won't be able to build a regular lattice based on a single regular polygon [Wiki] if each point has exactly 5 or exactly 6 (or more) nearest neighbors. But many semi-regular lattices also called tilings [Wiki], such as square-hexagonal [Wiki], exist. This also illustrates the fact that lattices form a group [Wiki], where shifting (also called translation) corresponds to the addition operation, and stretching is the scalar multiplication [Wiki]. Each shift vector uniquely characterizes a lattice, and the other way around. Also, an infinite 2-D lattice shifted by the vector $(\mu, \mu') = (h/\lambda, k/\lambda)$, regardless of $h, k \in \mathbb{Z}$, is topologically unchanged. The two lattices are congruent to each other modulo $1/\lambda$, in the same sense that (in one dimension) the numbers 30.628 and 40.052 are congruent [Wiki] to each other modulo 2.356 (in the latter case, because $30.628 - 40.052 = -4 \times 2.356$ is a multiple of 2.356).

Exercise 20 [MS] Nearest neighbors and size distribution of connected components**. Simulate 10 realizations of a stationary Poisson process of intensity $\lambda = 1$, each with $n = 10^3$ points distributed over a square window. Identify the connected components [Wiki] and their size (the number of points in each connected component). The purpose of the exercise is to study the distribution of the size, denoted as S. In particular, what is the proportion of connected components with only 2 points ($P[S = 2]$), 3 points ($P[S = 3]$) and so on? For connected components, use the undirected graph, that is: points V_i, V_j (also called vertices) are connected if V_i is nearest neighbor to V_j, or the other way around. The questions are:

- Estimate the probabilities in question via simulations. When computing the proportions using multiple realizations of the same process, do we get a similar empirical distribution for S, across all realizations? Does the empirical distribution seem to convergence, when increasing n, say from $n = 10^3$ to $n = 10^4$ or $n = 10^5$?
- Do the same experiment with a Poisson-binomial process, with $\lambda = 1$ and $s = 0.15$. Do we get the same distribution for S? What about $P[S = 2]$?
- Generate a particular type of random graph, called random NN graph, as follows. Let $V_1, \dots, V_n$ be the n vertices of the graph (their locations do not matter). For the "nearest neighbor" to vertex V_k ($k = 1, \dots, n$), randomly pick up one of the n vertices except V_k itself. Two points (vertices) can have the same nearest neighbor. Now study the distribution of S via simulations. Is it the same as for the graph generated by the nearest neighbors in a stationary Poisson point process?
- This is the most difficult part. Let $P(S = k), k = 2, 3, \dots$ be the size distribution for connected components of a stationary Poisson process; S is a random variable. Of course, it does not depend on λ. Does it uniquely characterize the Poisson process, in the same way that the exponential distribution for interarrival times uniquely characterizes the Poisson process in one dimension? Do we have $P(S = 2) = \frac{1}{2}$, not only for Poisson processes, but also for a much larger class of point processes?

Useful references about random graphs [Wiki] include "The Probabilistic Method" by Alon and Spencer [1] (available online here), and "Random Graphs and Complex Networks" by Hofstad [77] (available online here). See also here.

Hints

Beware of the boundary effect; to minimize the impact, use a uniform distribution for F (the distribution attached to the points of the Poisson-binomial process) and $n > 10^3$. When the scaling factor s is zero, there is only one connected component of infinite size ($P[S = \infty] = 1$): this is a singularity, as illustrated on the left plot in Figure 2. But as soon as $s > 0$, all the connected components are of finite size and rather small. The smallest ones have two points as each point has a nearest neighbor, thus $P[S < 2] = 0$. When $s = \infty$, the process becomes a stationary Poisson process, see Theorem 4.5.

I conjecture that stationary Poisson processes and some other (if not all) Poisson-binomial processes share the exact same discrete probability distribution for the size of connected components defined by nearest neighbors, and abbreviated as CCS distribution. Thus, unlike the point count or nearest neighbor distance distributions, the CCS distribution can not be used to characterize a Poisson process. For random graphs, the CCS distribution is different from that of a Poisson process. I used a Kolmogorov-Smirnov test [Wiki] (see also [26] available online here) to compare the two empirical CCS distributions – the one attached to Poisson processes versus the one attached to random NN graphs – and concluded, based on my sample size ($n = 10^4$ points or vertices), that they were statistically different.

To conclude, it appears that the CCS distribution can not be arbitrary. Many point processes seem to have the same CCS distribution, called attractor distribution, and these processes constitute the domain of attraction of the attractor. The concepts of domain of attraction and attractor is used in other contexts such as dynamical systems [Wiki] or extreme value theory [Wiki] (also, see [7] page 317). The most well known analogy is the Central Limit Theorem, where the Gaussian distribution is the main attractor, and the Cauchy distribution is another one. In chapter 11 of "The Probabilistic Method" [1], dealing with the size of connected components in random graphs, the author introduces a random variable T_c, also counting a number of vertices (called nodes in the book). Its distribution has all the hallmarks of an attractor. See Theorem 11.4.2 (page 202) in the book in question.

To find the connected components, you can use the source code in Section 6.5. To simulate point processes, you can use the source code in Section 6.4: it produces an output file `PB_NN_dist_full.txt` that can be used as input, without any change, to the connected components algorithm in Section 6.5. Exercise 21 features a similar problem, dealing with cliques rather than connected components.

Exercise 21 [M] Maximum clique problem. In undirected graphs [Wiki], a clique is a set of vertices (also called nodes) all connected to each other. In nearest neighbor graphs, two points are connected if one of them is a closest neighbor to the other one. How would you identify a clique of maximum size in such a graph? No need to design an algorithm from scratch; instead, search the literature. Finding the maximum clique [Wiki] is NP-hard [Wiki], and the problem is related to the "P versus NP" conjecture [Wiki]. The maximum clique problem has many applications, in particular in social networks. Probabilistic properties of cliques in random graphs are discussed in "Cliques in random graphs" [8] (available online here) and "On the evolution of random graphs" [25] (available online here). See also [Wiki]. More recent articles include [30, 57], respectively available here and here.

Solution

In two dimensions, in an undirected nearest neighbor graph, the minimum size of a maximum clique is 2 (as each point has a nearest neighbor), and the maximum size is 3. A maximum clique must be a connected component. See definition of connected component in Exercise 20. If each point has exactly one nearest neighbor, then a connected component of size $n > 1$ has n or $n-1$ edges (the arrows on the right plot in Figure 2), while a clique of size n has exactly $\frac{1}{2}n(n-1)$ edges. This is why maximum cliques of size larger than 3 don't exist. But in d dimensions, a maximum clique can be of size $d+1$. The maximum clique can be found using the MaxCliqueDyn algorithm [Wiki].

5.5 Miscellaneous

This section features problems that don't fit well in any of the previous categories.

Exercise 22 [M] Computing moments using the CDF. The purpose is to prove a formula to compute the moments of a random variable, using the cumulative distribution function (CDF), rather than the density. If X is a univariate random variable with CDF $F(x) = P(X < x)$, and r is a positive integer, prove the following:

- If X is positive, then $\text{E}[X^r] = r\int_0^\infty x^{r-1}(1-F(x))dx$
- If X is symmetric around the origin and r is even, then $\text{E}[X^r] = 2r\int_0^\infty x^{r-1}(1-F(x))dx$. If r is odd, $\text{E}[X^r] = 0$.

Solution A solution for the general case, or when X is positive, can be found here. If X is symmetric around the origin, then $F(x) = 1 - F(-x)$, and the result follows easily.

Exercise 23 [S] Simulations: generalized logistic distribution. Implement a routine that generates deviates for the generalized logistic distribution, using the quantile function $Q(u)$ in Formula (12), with a uniform distribution on $[0, 1]$ for u. Do the same for the Laplace distribution defined in Section 1.1. Simulate 1-D and 2-D Poisson-binomial point processes, using a Laplace and generalized logistic distribution for F. For the generalized logistic distribution, try different values for the parameters $\alpha, \beta, \tau, \mu, \lambda$.

Hint

Use inverse transform sampling to simulate Laplace deviates. That is, use the Laplace quantile function $Q(u)$ with uniform deviates on [0,1] for u; $Q(u)$ is the inverse of the Laplace cumulative distribution function F listed in Section 1.1.

Exercise 24 [S] Riemann Hypothesis. Refer to Section 2.3.2 for the material and notations discussed here. The hole in Figure 8, on the top left plot corresponding to $\sigma = 0.75$ and $s = 0$, is observed when $0 \leq t \leq 200$. Try other intervals, say $[t, t+\tau]$, for much larger values of t and (say) $\tau = 200$. See if the hole gets any smaller. Try $s = 10^{-2}$, instead of $s = 10^{-3}$ in Formula (20) and (21): now the hole is entirely gone. This shows how sensitive the η function is to small perturbations. Finally, find the first 40 values $t = t_1, \dots, t_{40}$, with $t > 0$, solutions of $\Im[\eta(\sigma+it)] = 0$, when $\sigma = \frac{1}{2}$, using numerical techniques. How many of these roots are also solution to $\Re[\eta(\sigma+it)] = 0$? Such values of t correspond to the non-trivial complex zeros of the Riemann zeta function, on the critical line $\sigma = \frac{1}{2}$.

Solution

The challenge here is the slow and chaotic convergence of the two series (real and imaginary parts) representing the function $\eta(\sigma+it)$ in Formula (18) and (19). I refer to t as the time. The larger t, the smaller the time increments required to correctly plot the orbit. These increments can be as small as 0.01 if $t \approx 10^3$, to not miss any rare value, say t_0, resulting in $\eta(\sigma+it_0)$ unusually close to the origin when $\sigma = 0.75$. A convergence acceleration technique is described in Exercise 25.

Exercise 25 [S*] Convergence acceleration. Design a basic algorithm for convergence acceleration of alternating series [Wiki]. How does it perform, when computing the sum in Formula (19)? Try with $s = 0.75$ and $t = 18265.2$ (the correct value of the sum is about 0.292040897 if you ignore the sign, see Mathematica computation here).

Solution

If $S_n = a_1 + a_2 + \cdots + a_n$ converges to S, and the a_k's are alternating, then one can proceed as follows:

- Let $S'_n = a'_1 + a'_2 + \cdots + a'_n$ with $a'_k = \alpha a_k + (1-\alpha)a_{k+1}$, and $0 \leq \alpha \leq 1$ chosen to maximize the speed of convergence of S'_n.
- Let $S''_n = a''_1 + a''_2 + \cdots + a''_n$ where $a''_k = \alpha' a'_k + (1-\alpha')a'_{k+1}$, and $0 \leq \alpha' \leq 1$ chosen to maximize the speed of convergence of S''_n.

One can continue iteratively with S'''_n and so forth, each new sum converging faster to S than the previous one. Also, the sequence a_1, a'_1, a''_1 and so on, rapidly converges to S. However, it fails to work in our example.

The reason is because, despite the appearance, the series in Formula (19) is not an alternating one. Indeed, hundreds, and even trillions of trillions of consecutive terms, depending on t, can have the same sign despite the $(-1)^k$ factor attached to each term. This behavior creates numerical instability. The explanation is as follows: If for some large k in Formula (18) or (19), the quantity $t \log(k+1) - t \log k \approx t/k$ is close to an odd multiple of π, then around that k, a lot of terms in the series will have the same sign and similar value (as opposed to the regular alternating behavior). As a result, if k is not large enough (but not too small) when this happens, a sum that seems to have converged, will suddenly experience a huge shift. This is what happens here, most strikingly when $k = 5814$ and $t = 18265.2$, leading to $t/k = 3.141589\ldots$ very close to π, and resulting in the odd behavior around $k = 5814$, illustrated in Figure 24. The X-axis represents k and the Y-axis represents the value of the partial sum computed using k terms.

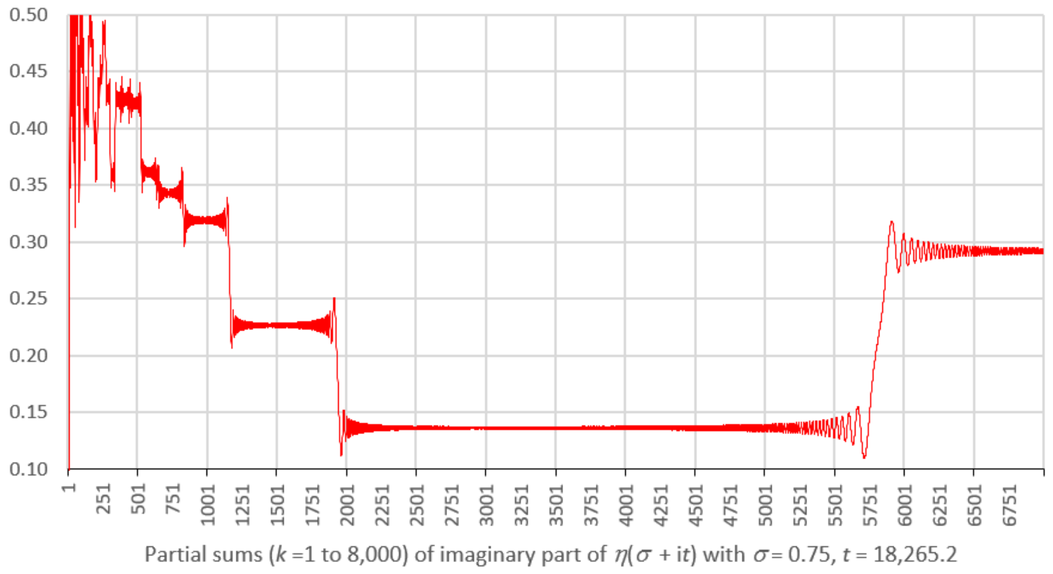

Figure 24: Chaotic convergence of partial sums in Formula (19)

There are various workarounds to deal with this issue. First, the Dirichlet eta function η has numerous representations: you can choose one that is more suitable for computation purposes. But even if you want to stick to Formula (19), you can improve it by splitting the sum into two parts:

- One part that deals with the few dips and spikes, easy to identify. Here, the last one occurs at $k = 5814$.
- The second part is to compute the first few hundred terms by traditional means.
- Then combine both parts to get a good approximation of the final sum, in the end using much fewer operations than brute force, and having a good sense as to when convergence is reached.

To prove the convergence of the series in Formulas (18) and (19) representing the Dirichlet eta function, one can use the Dirichlet test [Wiki]. Note that without the factor $(-1)^k$ in Formulas (18) and (19), the series may not converge.

Exercise 26 **[S] Fast image filtering algorithm**. The filtering algorithm described in Section 3.4.3 requires a large moving window of 21×21 pixels, around each pixel. The size of this window is the bottleneck. How can you make the algorithm about 20 times faster, still keeping the same window size?

Solution

When filtering the image, the window used at (x, y), and the next one at $(x+1, y)$, both have $21 \times 21 = 441$ pixels, but these two windows have $441 - (2 \times 21) = 399$ pixels in common. So rather than visiting 441 pixels each time, the overlapping pixels can be kept in a 21×21 buffer. To update the buffer after visiting a pixel and moving to the next one to the right, one only has to update 21 values in the buffer: overwrite the column corresponding to the old 21 leftmost pixels, by the values derived from the new 21 rightmost pixels.

Exercise 27 **[M**] Confidence regions: theory and computations**. What are the foundations justifying the methodology used to build the confidence regions in Section 3.1.1? In particular, how would you proceed to find the values of $\sigma_p, \sigma_q, \rho_{p,q}$ in Formula (27)? Does G_γ depend on n, p or q? Why not? What justifies the choice of an ellipse for the confidence region? What is the role of Hotteling's distribution in the methodology? How would you tabulate G_γ via simulations? Can you think of a different type of confidence region?

The goal here is to identify references that answer the questions, rather than trying to prove everything on your own. There is a considerable amount of tightly packed material in Section 3.1.1, presented at a high level. I only ask you, in this exercise, to dig just one level beneath the surface.

Solution

Each of the $2n$ observations is realization of a Bernoulli random vector $(U_k, V_k) \in \{(0,0),(0,1),(1,0)\}$, with $k = -n, \dots, n-1$. In particular, $U_k = 1$ if the interval B_k defined by Formula (25) contains exactly one point of the Poisson-binomial process, otherwise $U_k = 0$. Likewise $V_k = 1$ if B_k contains exactly two points, otherwise $V_k = 0$. The statistic p (a random variable depending on n) is the proportion of 1 in the sequence (U_k), and q is the proportion of 1 in (V_k). From this, it follows that $\sigma_p = \sqrt{p(1-p)}$, $\sigma_q = \sqrt{q(1-q)}$, $p + q \leq 1$, and U_k, V_k are negatively correlated. The proportions of $(0,0),(1,0)$ and $(0,1)$ among the (U_k, V_k) are respectively $1-(p+q), p$ and q. From this, it follows that $\rho_{p,q} = -pq/\sqrt{pq(1-p)(1-q)}$.

If the random vectors (U_k, V_k) were identically and independently distributed (iid), things would be easier thanks to the multivariate central limit theorem [Wiki], despite the strong correlation between U_k and V_k. Unfortunately, they are neither. Exercise 9 shows that the point counts are not identically distributed in general, and Exercise 10 shows the lack of independence. But the dependencies are local and very weak. Also a careful choice of non-overlapping B_k's in Formula (25) – inspired by Theorem 4.1 – makes the point counts almost identically distributed. They are in fact asymptotically iid. The length of B_k, set by Formula (24), is very well approximated by (and asymptotically equal to) $1/\lambda$, to minimize any problem. Section 3.1.2 provides further reassurance. Finally, when n is large, the B_k's are *on average* far away from each other, further dampening dependencies and related issues. And as a bonus, the bias caused by boundary effects tends to zero. By asymptotically, I mean when $n \to \infty$.

In the remaining of this discussion, I consider the previous issue as overcome. I proceed as if the (U_k, V_k) were iid. Thus, we can use the central limit theorem (CLT) *as is*. If we only had one statistic p and $2n$ observations, then the CLT states that $Z = \sqrt{2n} \cdot (p - \mu_p)/\sigma_p \to \mathcal{N}(0,1)$ as $n \to \infty$. In two dimensions, σ_p^2 is replaced by the 2×2 symmetric covariance matrix, denoted as Σ or $\Sigma_{p,q}$. Its inverse (the analogous of σ_p^{-1} in one dimension), is denoted as Σ^{-1}. The multivariate CLT implies that $Z = \sqrt{2n} \cdot (p - \mu_p, q - \mu_q)\Sigma^{-1/2} \to \mathcal{N}(0, I)$. That is, we have convergence to a bivariate normal distribution [Wiki] (also called multivariate Gaussian) of zero mean and identity covariance matrix I [Wiki]. Also,

$$\Sigma^{-1} = \frac{1}{1-\rho_{p,q}^2} \cdot \begin{bmatrix} \sigma_p^{-2} & \rho_{p,q}\sigma_p^{-1}\sigma_q^{-1} \\ -\rho_{p,q}\sigma_p^{-1}\sigma_q^{-1} & \sigma_q^{-2} \end{bmatrix}$$

In one dimension, Z^2 has a chi-squared distribution with one degree of freedom at the limit as $n \to \infty$. The Berry-Esseen theorem [Wiki] quantifies the stochastic error (that is, the quality of the approximation) when n is not infinite. In two dimensions, Z^2 is replaced by

$$\begin{aligned} Z \cdot Z^t &= \Big[\sqrt{2n}\ (p-\mu_p, q-\mu_q)\ \Sigma^{-1/2}\Big] \times \Big[\sqrt{2n}\ (p-\mu_p, q-\mu_q)\ \Sigma^{-1/2}\Big]^t \\ &= 2n\ (p-\mu_p, q-\mu_q)\ \Sigma^{-1}(p-\mu_p, q-\mu_q)^t \\ &= \frac{2n}{1-\rho_{p,q}^2} \cdot \left[\Big(\frac{p-\mu_p}{\sigma_p}\Big)^2 - 2\rho_{p,q}\Big(\frac{p-\mu_p}{\sigma_p}\Big)\Big(\frac{q-\mu_q}{\sigma_q}\Big) + \Big(\frac{q-\mu_q}{\sigma_q}\Big)^2\right] \end{aligned}$$

and still has a chi-squared distribution [Wiki], but this time with two degrees of freedom [Wiki]. This explains the choice of $H_n(x,y,p,q)$ in Formula (27), and why G_γ does not depend on p, q and quickly converges as $n \to \infty$. Here the symbol ^{t} denotes the transposition operator [Wiki], transforming a row vector into a column vector. Also, $Z^{-1/2} \cdot (Z^{-1/2})^t = Z^{-1}$. The chi-squared limit is a particular case of Cochran's theorem [Wiki]. It assumes that the exact values of $\mu_p, \mu_q, \sigma_p, \sigma_q, \rho_{p,q}$ are known. Unfortunately, this is not the case here: these values are replaced by their estimates based on p and q. As a result, in two dimensions, the chi-squared must be replaced by Hotteling's distribution [Wiki]. The proof of these results (the fact that the square of a Gaussian is a chi-squared, and so on) is based on the characteristic functions of these distributions.

The ellipse is the best possible shape for the confidence region: given a confidence level γ, it is the one of minimum area. To see why, start building a tiny, almost empty confidence region with $\gamma \approx 0$. You need to start at the maximum of the density function (here, a bivariate Gaussian by virtue of the central limit theorem). As you increase γ, the confidence region expands. But to keep it expanding at the slowest possible rate (keeping its area minimum at all times), you need to follow the contour lines of the density: the curves where the density is constant. For the Gaussian distribution, these contour lines are ellipses. But the same principle is true for any continuous bivariate density. In general, the shape will not be an ellipse.

There is an alternative definition of confidence region, called dual confidence region, leading to non-elliptic shapes even for Gaussian distributions. It consists of computing the confidence region for all (p, q) in the proxy space, rather than for your estimate (p_0, q_0) only. If the confidence region of some (p, q) contains (p_0, q_0), then it is part of (p_0, q_0)'s newly defined confidence region. The boundary of the newly defined confidence region of (p_0, q_0) consists of the points (x, y) satisfying

$$\frac{2n}{1-\rho_{x,y}^2} \cdot \left[\left(\frac{p-x}{\sigma_x}\right)^2 - 2\rho_{x,y}\left(\frac{p-x}{\sigma_x}\right)\left(\frac{q-y}{\sigma_y}\right) + \left(\frac{q-y}{\sigma_y}\right)^2\right] = H_\gamma, \tag{45}$$

with (p, q) set to (p_0, q_0). Compare Formula (27) with (45). Clearly, the latter does not correspond to the equation of an ellipse; the former does. The roles of (p, q) and (x, y) have been swapped. Also note the use of a different "scale" H_γ instead of G_γ. Yet in practice, the two methods yield almost identical results. Both the standard and newly defined confidence regions are implemented in the spreadsheet. An example using the standard region is featured on the left plot in Figure 11. Tables for G_γ and H_γ are provided in the `Confidence_Region` tab in the spreadsheet in question (`PB_Independence.xlsx`): see columns `F`, `G`, and `K`. I produced them via simulations, based on the code in column `Y`.

Last but not least, in the end the goal is to obtain confidence regions for the parameter (λ, s) in the parameter space, not for the proxy vector (p, q) in the proxy space . The final step consists of using the inverse mapping defined in Section 3.1.1, to map the confidence region built in the proxy space, onto the parameter space. The challenge here is to prove that the mapping is one-to-one. This is still an open question. Most likely, the final confidence region in the parameter space won't be an ellipse. There is an easy formula to do the mapping from the parameter space to the proxy space, see Section 3.1.2. But the inverse mapping, needed here, is a bit less easy to perform.

Exercise 28 [M*] Minimum set covering 90% of a distribution. This is related to the confidence regions discussed in Exercise 27. It consists of (1) finding the shape of the 2D set of minimum area, covering a proportion γ of the mass of a specific 2D probability distribution, and (2) determining its area.

Solution

Let S_γ be the set in question, and $f(x, y)$ be the density attached to the distribution. I assume that the density has one maximum only, and that it is continuous everywhere on $\mathbb{R}^2$. Thus the problem consists of finding the set S_γ of minimum area, such that

$$\int\int_{S_\gamma} f(x, y)dxdy = \gamma. \tag{46}$$

It is easy to see that the boundary of S_γ is a contour line of $f(x, y)$. To build S_γ, you start at the maximum of the density, and to keep the area minimum, the set must progressively be expanded, strictly following contour lines, until (46) is satisfied. So

$$S_\gamma = \{(x, y) \in \mathbb{R}^2 \text{ such that } f(x, y) \leq G_\gamma\},$$

where G_γ must be chosen so that (46) is satisfied. Assuming $\max f(x, y) = M$, the volume covered by S_γ is

$$\gamma = z_\gamma \cdot |S_\gamma| + \int_{z_\gamma}^{M} |R(z)|dz, \tag{47}$$

where $R(z) = \{(x, y) \in \mathbb{R}^2 \text{ such that } f(x, y) = z\}$, and $|\cdot|$ denotes the area of a 2D domain. Clearly, $|S_\gamma| = |R(z_\gamma)|$. So there is only one unknown in Equation (47), namely z_γ. Finally, $G_\gamma = z_\gamma$, and thus the value of G_γ is found by solving (47). The area of S_γ is thus $|S_\gamma| = |R(G_\gamma)|$.

6 Source Code, Data, Videos, and Excel Spreadsheets

My source code is available online at github.com/VincentGranville/Point-Processes, as well as in this textbook. It is written using only basic data structures and manipulations available in all programming languages, such as strings, arrays, stacks, subroutines, regular expressions and hash tables, to make it easy to read and rewrite in Java, C++ or other languages. The visualizations are performed either in R with the Cairo graphics library [Wiki] to create better scatterplots, or Python with the Pillow library to create PNG images pixel by pixel, including density estimation and clustering via image filtering.

My source code is designed to bring as much educational value as possible, without jeopardizing efficiency. It includes algorithms useful in many other contexts, such as the generation of random deviates from a logistic, Cauchy or Laplace distribution, and a fast, compact algorithm to detect connected components in a graph. The textbook version has detailed explanations. The source code repository is organized according to Table 8; to access the code online, click on the filename:

Filename	Textbook code	Purpose	Language
`PB_main.py`	Section 6.2	one-dimensional stats	Python
`PB_radial.py`	Section 6.3	radial clusters	Python
`PB_NN.py`	Section 6.4	NN distances	Python
`PB_NN_graph.py`	Section 6.5	NN connected components	Python
`PB_NN_arrows.r`	Section 6.6.1	NN graph visualization	R
`GD_util.py`	Section 6.6.2	maps creation (PNG images)	Python
`av_demo.r`	Section 6.7.1	video – Dirichlet eta function	R
`PB_clustering_video.py`	Section 6.7.2	video frames – fractal clustering	Python

Table 8: List of programs – NN stands for nearest neighbors, PB for Poisson-binomial

Below is a short description.

- `PB_main.py`: Computes the expectation, variance, and $P[N(B) = 0]$ of the point count $N(B)$ for any interval $B = [a, b]$, using respectively Formulas (4), (5) and (6), for various F's (uniform, logistic, Cauchy). Also computes the expectation, variance and higher moments $E[T^r]$ $(r > 0)$ of the interarrival times, using simulations. The main parameters are the scaling factor s, and the intensity λ .
- `PB_radial.py`: Generates a realization of a radial cluster process as described in Section 2.1. Used to produce Figures 3, 4, 15, and 16.
- `PB_NN.py`: Generates points and computes nearest neighbor distances and related statistics, for a realization of a superimposition of m shifted stretched Poisson-binomial processes (also called m-interlacing), as defined in Section 1.5.3, using Formulas 8 and 9 for the simulation of each individual point process. The output data consists of text files with tab-separated columns; they are used to study the distribution of nearest neighbor distances and statistical testing, and as input files for `PB_NN_arrows.r`, `PB_NN_graph.py` and `GD_util.py`.
- `PB_NN_arrows.r`: Based on output data from `PB_NN.py`, produces an image featuring the nearest neighbor points across m superimposed point processes (or a mixture of point processes), as defined in Section 1.5.3. Each arrow points from a point of the process, to its nearest neighbor(s). The color attached to each point indicates the process it belongs to, among the m processes used to generate the superimposition. See Figure 2.
- `PB_NN_graph.py`: Creates the list of all connected components of an undirected graph, for instance the nearest neighbor graph of a point process. Two points are considered connected if one of the two points is the nearest neighbor to the other one.
- `GD_util.py`: Small, easy-to-use home-made graphics library consisting of one function `GD_Maps`, relying on the Pillow library, to produce PNG images (bitmaps). Produces density and cluster maps such as Figure 19, as an alternative to traditional contour plots [Wiki], using image filtering and enhancing techniques including equalization. This library is used in `PB_NN.py`, at the very end.
- `av_demo.r`: Creates the video for the Dirichlet eta function, showing convergence of its series in the complex plane. Uses input data from `PB_inference.xlsx` to produce the video `av_demo2c.mp4`. See Section 2.4.1.

- PB_clustering_video.py: Generates the frames for the video imgPB.mp4 featuring fractal supervised clustering. See Section 2.4.2.

Detailed descriptions are included in the relevant subsections. Table 9 lists the data sets produced by the various programs, as well as the interactions between these programs. All these files are standard text files, with tab-separated columns. They are also available on my GitHub repository: click on the filename to find an example of the corresponding data set. The fields attached to each data set are described in the section covering the source code that produces it: for instance, Section 6.4 for the data set PB_NN_dist_full.txt, produced by PB_NN.py.

Filename	Output of	Input for	Description
PB_main.txt	PB_main.py	PB_inference.xlsx	one-dimensional stats
PB_radial.txt	PB_radial.py	PB_inference.xlsx	points of radial process
PB_cc.txt	PB_NN_graph.py		connected components
PB_r.txt	PB_NN.py	PB_NN_arrows.r	nearest neighbor graph
PB_NN.txt	PB_NN.py	PB_inference.xlsx	points of the process
PB_NN_mod.txt	PB_NN.py	PB_inference.xlsx	points of the process (mod $\frac{2}{\lambda}$)
PB_NN_dist_full.txt	PB_NN.py	PB_NN_graph.py	nearest neighbor distances
PB_NN_dist_small.txt	PB_NN.py	PB_inference.xlsx	nearest neighbor distances
PB-map.PNG	GB_util.py	Figure 19	density and cluster map
PB-hexa.png	PB_NN_arrows.r	Figure 2	nearest neighbor graph
av_demo2c.mp4	av_demo.r		video, Dirichlet eta function

Table 9: Source code architecture: input/output data flow between modules

The spreadsheets accompanying this textbook are discussed in Section 6.1. They are also accessible from the same GitHub repository, here.

6.1 Interactive Spreadsheets and Videos

Here I provide a brief overview of the spreadsheets and data animations (MP4 videos) referenced in the textbook. Most of the figures come from these documents. The content (columns, cells, formulas, graphs) is documented in details in the relevant material in the textbook. In Table 10, I provide references to the sections where the corresponding material is discussed, for each tab of the two spreadsheets.

General guidelines:

- Parameters highlighted in light yellow can be fine-tuned. Any change will be immediately visible on the graphs that are included in the same tab.
- Do not change the other parameters.
- Data produced directly or indirectly with the RAND() Excel function is updated each time you save the spreadsheet or change some parameters. New random deviates are automatically generated. This includes realizations of Poisson-binomial processes used for simulation purposes, and the resulting graphs.
- Most of the time, tabs are self-contained: all the required data (for instance, to generate a chart) is produced from within the same tab. On some occasions, raw data requiring of lot of storage is not provided. Instead, summary data is included in the spreadsheet. The source code to produce the summary data is in the same tab where it is used. This allows for full replication.
- The spreadsheets are available on my GitHub repository. It is best to download them rather than view them locally on my repository. GitHub, Google Drive and other sharing websites have poor Excel viewing capabilities. They may be good for basic spreadsheets, but mine include a lot of Excel features that are poorly rendered (if at all) on these platforms.
- The labels, headers and parameter names in the spreadsheet are compatible with those used in the textbook.
- I do not use macros, pivot tables, plug-ins, or other non-basic Excel features. A standard version of Excel 2013 (or above) is all you need.

The spreadsheets, PB_independence.xlsx and PB_inference.xlsx, are summarized in Table 10.

Spreadsheet	Tab	Description	Section
PB_independence	Periodicity	periodicity of point count expectations	3.1.2
PB_independence	Confidence_Region	standard and dual confidence regions	3.1.1
PB_independence	MC_Estimation	minimum contrast estimation	3.1.1
PB_independence	Summary	testing independence of point counts	3.1.3
PB_independence	Dataset_A, B, C	raw datasets used in Summary tab	3.1.3
PB_inference	N_k	expectation and variance of point count	3.2.1
PB_inference	E(T^2)	second moment of interarrival times	3.2.1
PB_inference	Rayleith_Test	model fitting, two dimensions	3.4.2
PB_inference	Elbow_Brownian	elbow rule, Figure 20	3.4.4
PB_inference	Elbow_Riemann	elbow rule, Figure 21	3.4.4
PB_inference	Video_Riemann	data animation, Dirichlet function	2.4

Table 10: Spreadsheet structure and references to textbook sections

Note About the Videos

The material about the videos (data animation, MP4 files) and how to create them, is described in Section 2.4.

6.2 Source Code: Point Count, Interarrival Times

On GitHub: PB_main.py. The code below is used to compute two sets of statistics:

- Point count: $\text{E}[N(B)], \text{Var}[N(B)]$ and $P[N(B) = 0]$, where $B = [a, b]$
- Interarrival times: $\text{E}[T], \text{Var}[T]$ and $\text{E}[T^r]$, with $T = T(\lambda, s)$

The input parameters are $a, b, r > 0$ and $\lambda > 0$. The code below tests various values of s, between $s = 0.05$ and $s \leq 0.40$, with increments equal to 0.2. For the point count, the theoretical Formulas (3), (4) and (5) are used, with the infinite sums truncated to k between $-n_1$ and n_1. The default value is $n_1 = 10^3$. For the interarrival times, simulations are used instead, by generating one instance (realization) of a Poisson-binomial process consisting of $n_2 = 3 \times 10^3$ points. Both n_1 and n_2 can be changed, but s should always be much smaller than n_2. The computation of the main statistics use a subset of the n_2 points, to minimize boundary effects.

Three options (Logistic, Cauchy and Uniform) are offered for the CDF (cumulative distribution function) F. These three options are stored in the model list. Two useful functions are provided: one computing the CDF (see Section 6.2.4), and one generating the corresponding deviates (see Section 6.2.3). The parameter type specifies which distribution F is used at any time. The main program performs a loop on s, with an inner loop on the three CDF. The output is stored in a text file named PB_main.txt. Table 3 is produced with this code.

The full program consists of all the pieces of code in Section 6.2, in the same order. You also need to add one instruction at the very bottom: main(). The reason is because in Python, functions must be defined above the main code that calls them. A workaround is to define the main code as a function, listed above all the other functions. This is what I did here: the main code is in the main() function, which must be called after all the other functions have been defined. This is also how it is implemented in the GitHub version.

```python
# PB_main.py

import math
import random
random.seed(100)

model=("Uniform","Logistic","Cauchy")
pi= 3.14159265358979323846264433
seed=4565 # allows for replicability (to produce same random numbers each time)

llambda=1 # represents lambda [lambda is reserved keyword in Python]
aa =-0.75 # B=[aa, bb] is the interval used to compute exact Expectation and var
```

```
bb = 0.75 # see aa
r = 0.50 # to compute E[T^r], r>0
         # E[T^r] tends to r!/(lambda)^r as s tends to infinity
n1 = 10000 # compute E[N(B)], Var[N(B)]: k between -n1 and +n1
         # n1 much larger than s (if F has thick tail)
         # reduce n1 if program too slow [speed ~ O(n1 log n1
n2 = 30000 # Simulation: Xk with index k between -n2 and +n2

#---
def main():

  OUT=open('PB_main.txt',"w") # computations saved in file pb.txt
  line = "Type\tlambda\ts\ta\tb\tr\tE[N]\tVar[N]\tP[N=0]\t";
  line = line+"E[T]\tVar[T]\tE[T^r]\n";
  OUT.write(line)

  for type in model:
    s=0.05
    while s <= 40:
      line=str(type)+"\t"+str(llambda)+"\t"+str(s)+"\t"+str(aa)+"\t"+str(bb)+"\t"+str(r)+"\t"
      print("F = ",type," | lambda = ",llambda," | s=",s) # show progress on the screen

      # Compute E[N(B)], Var[(B)], P[B=0] via formula
      (exp,var,prod)=E_and_Var_N(type,llambda,s,aa,bb,n1)
      line=line+str(exp)+"\t"+str(var)+"\t"+str(prod)+"\t"

      # Compute E[T], Var[T] via simulations
      random.seed(seed) # to produce same random deviates each time (for replicability)
      (exp,var,moment)=var_T(type,llambda,s,r,n2)
      line=line+str(exp)+"\t"+str(var)+"\t"+str(moment)+"\n"

      OUT.write(line)
      s=s+0.2

  OUT.close()
```

6.2.1 Compute $\mathbf{E}[N(B)]$, $\mathbf{Var}[N(B)]$ and $P[N(B) = 0]$

This computation is done using the exact Formulas (3), (4) and (5). The input parameters are λ, s, a, b, with $B = [a, b]$. See code below.

```
def E_and_Var_N(type,llambda,s,aa,bb,n):

  # Return E[N(B)], Var[N(B)] and P[N(B)=0] with B=[aa, bb]
  # expectation -> E[N(B)]
  # variance -> Var[N(B)]
  # product  -> P[N(B)=0]
  # Type specifies the distribution F, lambda the intensity, s the scaling factor

  variance=0
  expectation=0
  product=0
  flag=0

  for k in range(-n1,n1+1):
    f1=CDF(type,llambda,s,k,bb)
    f2=CDF(type,llambda,s,k,aa)
    if 1-f1+f2 == 0:
      flag=1
    else:
      product=product+math.log(1-f1+f2)
    expectation=expectation+(f1-f2)
    variance=variance+((f1-f2)*(1-f1+f2))
  if flag==1:
    product=0
  else:
    product=math.exp(product)
  return[expectation,variance,product]
```

6.2.2 Compute $\mathbf{E}[T]$, $\mathbf{Var}[T]$ and $\mathbf{E}[T^r]$

This computation is done via simulations. The input parameters are λ, s, r. See code below.

```
def var_T(type,llambda,s,r,n):

  # Return var(T) and E(T^r) computed on simulated data (2n+1 points)
  # Type specifies the distribution F, lambda the intensity, s the scaling factor
  # r=1 yields the expectation

  xs=[]
  m=0

  for k in range(-n,n+1):
    ranx=random.random()
    xs.append(deviate(type,llambda,s,k))
    m=m+1
  xs.sort()
  expectation=0
  variance=0
  moment_r=0
  k1=int(m/4)
  k2=int(3*m/4)
  for k in range(k1,k2+1):
    dist=(xs[k]-xs[k-1])
    expectation=expectation+dist
    variance=variance+(dist*dist)
    moment_r=moment_r+(dist**r)
  expectation=expectation/(k2-k1+1)
  variance=(variance/(k2-k1+1))-(expectation*expectation)
  moment_r=moment_r/(k2-k1+1)
  return[expectation,variance,moment_r]
```

6.2.3 Produce random deviates for various F's

Below is the code to generate deviates from selected distributions (uniform, logistic, Cauchy), using inverse transform sampling [Wiki]. Note that these deviates are centered at k, which is an input parameter. To produce standard deviates (centered at the origin), set k to zero. The scaling factor s is a function of the variance σ^2. Table 1 provides the conversion table between s and σ^2. A standard reference on this topic is Ripley [69]. See also [50].

```
def deviate(type,llambda,s,k):

  # Generate random deviate for F determined by type
  # centered at k/lambda, scaling factor s

  ranx=random.random()
  if type == "Logistic":
    z=k/llambda+s*math.log(ranx/(1-ranx))
  elif type == "Uniform":
    z=k/llambda+2*s*(ranx-1/2)
  elif type == "Cauchy":
    z=k/llambda+s*math.tan(pi*(ranx-1/2))
  return(z)
```

6.2.4 Compute $F(x)$ for Various F

Below is the code to compute $F(x)$, the value of the cumulative distribution (CDF) for selected distributions, given x, λ and s. Note that these CDF's are centered at k, which is an input parameter. To use for the standard

CDF (centered at the origin), set k to zero. The scaling factor s is a function of the variance σ^2. Table 1 provides the conversion table between s and σ^2.

```
def CDF(type,llambda,s,k,x):

  # Returns F((x-k/lambda)/s), with F determined by type

  if type == "Logistic":
    z= 1/2+ (1/2)*math.tanh((x-k/llambda)/(2*s))
  elif type == "Uniform":
    z= 1/2 + (x-k/llambda)/(2*s)
    if z<=0:
      z=0
    if z>1:
      z=1
  elif type == "Cauchy":
    z= 1/2 +math.atan((x-k/llambda)/s)/pi;
  return(z)
```

6.3 Source Code: Radial Cluster Simulation

On GitHub: PB_radial.py. Simulates realizations of the cluster processes featured in Sections 2.1 and 2.1.2, to produce Figures 3 and 4. The parent process is Poisson-binomial with F being uniform, $\lambda = 1$, and consisting of the blue crosses in Figure 4. The points of this process are called the centers. A radial process – the child process – with up to 15 points, is attached to each random center (X, Y). It shows up as green dots in Figure 4. Each point (X', Y') of this child process is generated as follows:

$$X' = X + \log\Big(\frac{U}{1-U}\Big)\cos(2\pi V)$$
$$Y' = Y + \log\Big(\frac{U}{1-U}\Big)\sin(2\pi V)$$

where U, V are independent uniform deviates on $[0, 1]$.

```
# PB_radial.py

import math
import random
random.seed(100)

s=10

pi=3.14159265358979323846264338

file=open('PB_radial.txt',"w")
for h in range(-30,31):
   for k in range(-30,31):

        # Create the center (parent Poisson-binomial process, F uniform)

        ranx=random.random()
        rany=random.random()
        x=h+2*s*(ranx-1/2)
        y=k+2*s*(rany-1/2)
        line=str(h)+"\t"+str(k)+"\tCenter\t"+str(x)+"\t"+str(y)+"\n"
        file.write(line)

       # Create the child, radial process (up to 15 points per center)

        M=int(15*random.random())

        for m in range(M):
           ran1=random.random()
           ran2=random.random()
           factor=math.log(ran2/(1-ran2))
           x1=x+factor*math.cos(2*pi*ran1);
           y1=y+factor*math.sin(2*pi*ran1);
           line=str(h)+"\t"+str(k)+"\tLocal\t"+str(x1)+"\t"+str(y1)+"\n"
           file.write(line)
 file.close()
```

6.4 Source Code: Nearest Neighbor Distances

On GitHub: PB_NN.py. Generates points and computes nearest neighbor distances and related statistics, for a realization of a superimposition of m shifted stretched Poisson-binomial processes (also called m-interlacing), as defined in Section 1.5.3, using Formulas 8 and 9 for the simulation of each individual point process. The output data consists of text files with tab-separated columns. They are used to study the distribution of nearest neighbor distances and statistical testing, and as input files for PB_NN_arrows.r, PB_NN_graph.py and GD_util.py.

The source code is divided into 5 parts.

Part 1 consists of initializing the following global variables:

- Nprocess: The number m of processes used to create the superimposed process.
- seed: To initialize the pseudo-random number generator, so that the same data is produced each time the program is run, for easy replication.
- s: The scaling factor s (note that $\lambda = 1$)
- ShiftX[i], ShiftY[i]: X- and Y-coordinates of the shift vector (arrays)
- StretchX[i], StretchY[i]: Stretching factors for the X and Y axes; set to 1 here.
- epsilon: Used for numerical stability.
- processID: Index of the point process (among the m point processes) currently accessed, in any loop.
- bitmap: 400×400 array to store and process an image in memory.
- string: Used for Excel compatibility to easily create scatterplots with multiple colors: one color for each processID. Should be replaced by a single TAB character if you don't use Excel; otherwise it consists of multiple TABs.

The fields of the output files are provided in Parts 2 to 5, where they are used.

```
# PB_NN.py
# lambda = 1

import numpy as np
import math
import random

# PART 1: Initialization

Nprocess=5           # number of processes in the process superimposition
s=0.15               # scaling factor
method=0     # method=0 is fastest
NNflag=False # set to True if you need to compute NN distances
window=10  # determines size of local filter [the bigger, the smoother]
nloop=3    # number of times the image is filtered [the bigger, the smoother]

epsilon=0.0000000001 # for numerical stability
seed=82431           # arbitrary number
random.seed(seed)    # initialize random generator

sep="\t"   # TAB character
shiftX=[]
shiftY=[]
stretchX=[]
stretchY=[]
a=[]
b=[]
process=[]
sstring=[] # string in Perl version
```

```
for i in range(Nprocess) :
  shiftX.append(random.random())
  shiftY.append(random.random())
  stretchX.append(1.0)
  stretchY.append(1.0)
  sstring.append(sep)
  # i TABs separating x and y coordinates in output file for points
  # originating from process i; Used to easily create a scatterplot in Excel
  # with a different color for each process.
  sep=sep + "\t"

processID=0
m=0 # number of points generated
height,width = (400, 400)

bitmap = [[255 for k in range(height)] for h in range(width)]
```

Part 2 generates a realization of m superimposed stretched shifted Poisson-binomial point processes, called m-interlacing; m is represented by the variable `Nprocess`. The index space is limited to $(h, k) \in \{-25, \dots, 25\} \times \{-25, \dots, 25\}$. The points of the process, along with their lattice index (h, k) and the individual process they belong to (`processID`), are saved in the output file `PB_NN.txt`. A subset of these points, those with coordinates in $[-20, 20] \times [20, 20]$, this time taken modulo $2/\lambda$ (with $\lambda = 1$), are saved in the `bitmap` array for further processing as well as in the output file `PB_NN_mod.txt`.

The restriction to a subset is to mitigate boundary effects. Taking the modulo allows you to magnify the patterns in the point distribution, to make statistical inference easier and to make the underlying shift-induced clustering structure visible to the naked eye. The modulo function is defined as follows: $x \mod \frac{2}{\lambda} = x - \frac{2}{\lambda}\lfloor x \cdot \frac{\lambda}{2} \rfloor$ where the brackets represent the integer function, also called floor function.

```
# PART 2: Generate point process, its modulo 2 version; save to bitmap and output files.

OUT = open("PB_NN.txt", "w")      # the points of the process
OUT2 = open("PB_NN_mod.txt", "w") # the same points modulo 2/lambda both in x and y
    directions

for h in range(-25,26):
   for k in range(-25,26):
      for processID in range(Nprocess):
         ranx=random.random()
         rany=random.random()
         x=shiftX[processID]+stretchX[processID]*h+s*math.log(ranx/(1-ranx))
         y=shiftY[processID]+stretchY[processID]*k+s*math.log(rany/(1-rany))
         a.append(x)  # x coordinate attached to point m
         b.append(y)  # y coordinate attached to point m
         process.append(processID) # processID attached to point m
         m=m+1
         line=str(processID)+"\t"+str(h)+"\t"+str(k)+"\t"+str(x)+sstring[processID]+str(y)+"\n"
         OUT.write(line)
         # replace sstring[processID] by \t if you don't care about Excel

         if x>-20 and x<20 and x>-20 and x<20:
            xmod=1+x-int(x)  # x modulo 2/lambda
            ymod=1+y-int(y)  # y modulo 2/lambda
            pixelX=int(width*xmod/2)
            pixelY=int(height*(2-ymod)/2) # pixel (0,0) at top left corner
            bitmap[pixelX][pixelY]=processID
            line=str(xmod)+sstring[processID]+str(ymod)+"\n"
            OUT2.write(line)
            # replace sstring[processID] by \t if you don't care about Excel
OUT2.close()
OUT.close()
```

Part 3 detects the nearest neighbor(s) to each point of the process, and compute the nearest neighbor distances. Only points in $[-20, 20] \times [20, 20]$ are considered, to mitigate boundary effects. The main loop is over all points of the process. Per convention, variables with the keyword "hash" in their name, represent hash tables. The output file `PB_NN_dist_small.txt` contains all that is needed to study the distribution of nearest neighbor distances for model-fitting purposes (see Section 3.4).

The output file `PB_NN_dist_full.txt` contains more fields, including the points and their nearest neighbor(s); this information is used to compute the connected components in the program `PB_NN_graph.py`. Here the variable `m` represents the number of points of the process. For each point `i`,

- `a[i]`, `b[i]` are the X and Y coordinate of point `i`.
- `NNidx[i]` is a nearest neighbor to point `i` (usually unique unless $s = 0$), and `NNx[i]`, `NNy[i]` are its X and Y coordinates.
- `mindist` is the distance between point `i` and its nearest neighbor point `NNidx[i]`.
- `NNidxHash[i]` is the list of points having `i` as nearest neighbor (separated by the character "~")

```
# PART 3: Find nearest neighbor points, and compute nearest neighbor distances.

if NNflag:

  OUT = open("PB_NN_dist_small.txt", "w") # the points of the process
  OUTf = open("PB_NN_dist_full.txt", "w") # the same points modulo 2/lambda both in x and
      y directions

  NNx=[]
  NNy=[]
  NNidx=[]
  NNidxHash={}

  for i in range(m):
    NNx.append(0.0)
    NNy.append(0.0)
    NNidx.append(-1)
    mindist=99999999
    flag=-1
    if a[i]>-20 and a[i]<20 and b[i]>-20 and b[i]<20:
      flag=0;
      for j in range(m):
        dist=math.sqrt((a[i]-a[j])**2 + (b[i]-b[j])**2) # taxicab distance faster to
            compute
        if dist<=mindist+epsilon and i!=j:
          NNx[i]=a[j] # x-coordinate of nearest neighbor of point i
          NNy[i]=b[j] # y-coordinate of nearest neighbor of point i
          NNidx[i]=j  # indicates that point j is nearest neighbor to point i
          # NNidxHash[i] is the list of points having point i as nearest neighbor;
          # these points are separated by "~" (usually only one point in NNidxHash[i]
          # unless the simulated points are exactly on a lattice, e.g. if s = 0)
          if abs(dist-mindist) < epsilon:
            NNidxHash[i]=NNidxHash[i]+"~"+str(j)
          else:
            NNidxHash[i]=str(j)
          mindist=dist
      if i % 100 == 0:
        print("Finding Nearest neighbors of point",i)
      line=str(i)+"\t"+str(mindist)+"\n"
      OUT.write(line)
      line=str(i)+"\t"+str(NNidx[i])+"\t"+str(NNidxHash[i])+"\t"+str(a[i])+"\t"
      line=line+str(b[i])+"\t"+str(NNx[i])+"\t"+str(NNy[i])+"\t"+str(mindist)+"\n"
      OUTf.write(line)

  OUTf.close()
  OUT.close()
```

Part 4 produces the output file `PB_r.txt` used by `PB_NN_arrows.r` to generate Figure 2. This file consists of the points of the process, with for each point idx: its X and Y coordinates a[idx], b[idx], its nearest neighbor point NNindex, the X and Y coordinates a[NNindex], b[NNindex] of point NNindex, and the individual process process[idx] that idx belongs to (for coloring purposes).

```
# PART 4: Produce data to use in R code that generates the nearest neighbors picture.

if NNflag:

  OUT = open("PB_r.txt","w")
  OUT.write("idx\tnNN\tNNindex\ta\tb\taNN\tbNN\tprocessID\tNNprocessID\n")

  for idx in NNidxHash:
    NNlist=NNidxHash[idx]
    list=NNlist.split("~")
    nelts=len(list)
    for n in range(nelts):
      NNindex=int(list[n])
      line=str(idx)+"\t"+str(n)+"\t"+str(NNindex)+"\t"+str(a[idx])+"\t"+str(b[idx])
      line=line+"\t"+str(a[NNindex])+"\t"+str(b[NNindex])+"\t"+str(process[idx])
      line=line+"\t"+str(process[NNindex])+"\n"
      OUT.write(line)

  OUT.close()
```

Part 5 consists of a single call the the function GD_Maps in GD_util.py (see Section 6.6.2) to produce two images: one representing the point density of the point process (to identify cluster centers, corresponding to the darkest gray level), and one representing (by a color) how each future, unobserved point should be classified based on its X and Y coordinates in the context of supervised clustering. See Figure 17 (original point process), and 19 after clustering / density estimation.

The X and Y coordinates are taken modulo $2/\lambda$; here $\lambda = 1$ (see Part 2 of this source code) and thus cover the entire, infinite 2-D space. The choice of the modulus (here $2/\lambda$, rather than $1/\lambda$) is dictated by the granularity of the underlying lattice space. The image is first processed in memory (the bitmap array) before being saved to PNG files (pb-cluster3.png and pb-density3.png). An high-pass (sharpening) kernel-based filter is applied nloop times to the entire bitmap image, using a $p \times p$ pixels filtering window. For a large image of fixed size, filtering the entire image once is $O(p^2)$ but can be reduced to $O(p)$ with a smarter implementation (see Exercise 26). The variable window represents p. See section 3.4 for details.

```
# PART 5: Creates density and cluster images.

img_cluster="PB-cluster" # use for output image filenames
img_density="PB-density" # use for output image filenames

from GD_util import *
GD_Maps(method,bitmap,Nprocess,window,nloop,height,width,img_cluster,img_density)
```

6.5 Source Code: Detection of Connected Components

On GitHub: PB_NN_graph.py. Creates the list of all connected components of an undirected graph, for instance the nearest neighbor graph of a point process. Two points are considered connected if one of the two points is the nearest neighbor to the other one. See Exercise 20.

The source code is divided into 3 parts.

Part1 reads the first two columns of PB_dist_full.txt produced by PP_NN.py (see Section 6.4). The first column represents the index idx of a point, and NNidx[idx] (in the second column) is the index of a point that has point idx as nearest neighbor.

Then, it creates the undirected graph hash, as follows: if a point with index k is nearest neighbor to a point with index idx, add point idx to hash[k], and add point k to hash[idx]. Thus hash[idx] contains all the points (their indices) directly connected to point idx; the points are separated by the character "~".

```
# PB_NN_graph.py
#
```

```
# Compute connected components of nearest neighbor graph
# Input file has two tab-separated columns: idx and idx2
#  idx is the index of of point, idx2 is the index of a nearest neighbor to idx
# Output file has two fields, for each principal component:
#  the list of points it is made up (separated by ~), and the number of points

# Example.

# Input:

# 100 101
# 100 103
# 101 100
# 101 102
# 103 100
# 103 102
# 102 101
# 102 100
# 102 103
# 102 104
# 104 102
# 106 105
# 105 107

# Output:

# ~100~103~102~104~101 5
# ~106~105~107 3

# PART 1: Initialization.

point=[]
NNIdx={}
idxHash={}

n=0
file=open('PB_dist_full.txt',"r") # input file
lines=file.readlines()
for aux in lines:
  idx =int(aux.split('\t')[0])
  idx2=int(aux.split('\t')[1])
  if idx in idxHash:
    idxHash[idx]=idxHash[idx]+1
  else:
    idxHash[idx]=1
  point.append(idx)
  NNIdx[idx]=idx2
  n=n+1
file.close()

hash={}
for i in range(n):
  idx=point[i]
  if idx in NNIdx:
    substring="~"+str(NNIdx[idx])
  string=""
  if idx in hash:
    string=str(hash[idx])
  if substring not in string:
    if idx in hash:
      hash[idx]=hash[idx]+substring
    else:
      hash[idx]=substring
  substring="~"+str(idx)
  if NNIdx[idx] in hash:
    string=hash[NNIdx[idx]]
  if substring not in string:
    if NNIdx[idx] in hash:
      hash[NNIdx[idx]]=hash[NNIdx[idx]]+substring
    else:
      hash[NNIdx[idx]]=substring
```

Part 2: Find the connected components. The algorithm is as follows. Browse the list of points. If a point `idx` has not yet been assigned to a connected component, create a new connected component `cliqueHash[idx]` containing `idx`; find the points connected to `idx`, add them to the stack (`stack`). Find the points connected to the points connected to `idx`, and so on recursively, until no more points can be added. Each time a point is added to `cliqueHash`, decrease the stack size by one. It takes about $2n$ steps to find all the connected components, where n is the number of points. This algorithm does not use recursive functions; it uses a stack instead, which emulates recursivity.

```
# PART 2: Find the connected components

i=0;
status={}
stack={}
onStack={}
cliqueHash={}

while i<n:

  while (i<n and point[i] in status and status[point[i]]==-1):
    # point[i] already assigned to a clique, move to next point
    i=i+1

  nstack=1
  if i<n:
    idx=point[i]
    stack[0]=idx; # initialize the point stack, by adding idx
    onStack[idx]=1;
    size=1 # size of the stack at any given time

    while nstack>0:
      idx=stack[nstack-1]
      if (idx not in status) or status[idx] != -1:
        status[idx]=-1 # idx considered processed
        if i<n:
          if point[i] in cliqueHash:
            cliqueHash[point[i]]=cliqueHash[point[i]]+"~"+str(idx)
          else:
            cliqueHash[point[i]]="~"+str(idx)
        nstack=nstack-1
        aux=hash[idx].split("~")
        aux.pop(0) # remove first (empty) element of aux
        for idx2 in aux:
          # loop over all points that have point idx as nearest neighbor
          idx2=int(idx2)
          if idx2 not in status or status[idx2] != -1:
            # add point idx2 on the stack if it is not there yet
            if idx2 not in onStack:
              stack[nstack]=idx2
              nstack=nstack+1
            onStack[idx2]=1
```

Part 3 saves the result to output text file `PB_cc.txt`. Each row corresponds to a connected component. The first column stores the connected component, as a string of point indices, separated by the character "~". The second column is the size (number of points) in the connected component in question.

```
# PART 3: Save results.
```

```
file=open('PB_cc.txt',"w")
for clique in cliqueHash:
  count=cliqueHash[clique].count('~')
  line=cliqueHash[clique]+"\t"+str(count)+"\n"
  file.write(line)
file.close()
```

6.6 Source Code: Visualizations, Density Maps

The code here produces the PNG images related to the clustering algorithms, and for Figure 2. Also, it is used to produce the PNG frames for some of the videos (MP4 files) in Section 6.7. This section requires some familiarity with basic image processing techniques such as color allocation, equalization or filtering, applied to clustering problems. The code still remains basic, and can be used as an introduction to image processing techniques for software engineers and scientists. It will teach you how to create an image pixel by pixel in some automated way, if you never tried before.

6.6.1 Visualizing the Nearest Neighbor Graph

On GitHub: PB_NN_arrows.r. Based on output data from PB_NN.py, produces an image showing the nearest neighbor points across m superimposed point processes (or a mixture of point processes), as defined in Section 1.5.3. Each arrow points from a point of the process, to its nearest neighbor(s). The color attached to each point indicates the process it belongs to, among the m processes used to generate the superimposition. See Figure 2.

The code below handles a superimposition of up to 5 point processes, but can easily be generalized to more than 5. Choosing many colors that are well contrasted and well rendered on the screen, is a science. It will be discussed in one of my upcoming books. The input file here is PB_r.txt, produced by PB_NN.py. Only a small window is displayed on the screen: $(x, y) \in [0, 5] \times [0, 5]$, to avoid cluttering. Try with $[-5, 5] \times [-5, 5]$, using the same input file and modifying the c() parameter accordingly in the plot function. The result is still nice.

My R code uses the Cairo graphics library [Wiki] to produce better, smoother graphics: this library uses anti-aliasing techniques [Wiki], making R plots look much nicer. For details, see here. The output image is PB_hexa2.png, integrated in Figure 2 in this textbook. The arrows function and its parameters are discussed here.

```
# install.packages('Cairo')
library('Cairo');
# CairoWin(6,6);
CairoPNG(filename = "c:/Users/vince/tex/PB-hexa2.png", width = 600, height = 600);

data<-read.table("c:/Users/vince/tex/PB_r.txt",header=TRUE);
a<-data$a; # x coordinate of point of the superimposed/mixture process
b<-data$b; # y coordinate of point of the superimposed/mixture process
aNN<-data$aNN; # x coordinate of nearest neighbor point to (a,b) across all processes
bNN<-data$bNN; # y coordinate of nearest neighbor point to (a,b) across all processes
processID<-data$processID;

plot(a,b,xlim=c(0,5),ylim=c(0,5),pch=20,cex=0,
               col=rgb(0,0,0),xlab="",ylab="",axes=TRUE );
arrows(a, b, aNN, bNN, length = 0.10, angle = 10, code = 2,col=rgb(0.7,0.7,0.7));

aa<-data$a[processID == 0];
bb<-data$b[processID == 0];
points(aa,bb,col=rgb(1,0,0),pch=20,cex=1.75);

aa<-data$a[processID == 1];
bb<-data$b[processID == 1];
points(aa,bb,col=rgb(0,0,1),pch=20,cex=1.55);

aa<-data$a[processID == 2];
bb<-data$b[processID == 2];
points(aa,bb,col=rgb(1,0.7,0),pch=20,cex=1.75);

aa<-data$a[processID == 3];
```

```
bb<-data$b[processID == 3];
points(aa,bb,col=rgb(0,0,0),pch=20,cex=1.75);

aa<-data$a[processID == 4];
bb<-data$b[processID == 4];
points(aa,bb,col=rgb(0,0.7,0),pch=20,cex=1.75);

dev.off();
```

6.6.2 Clustering and Density Estimation via Image Filtering

On GitHub: `GD_util.py`. Small, easy-to-use home made graphics library consisting of one function `GD_Maps`, relying on the Pillow library, to produce PNG images (bitmaps). Produces density and cluster maps such as Figure 19, as an alternative to traditional contour plots [Wiki], using image filtering and enhancing techniques including equalization. This library is used in `PB_NN.py`, at the very end.

Part 1 initializes the color palette for the cluster image. The input parameters of the function `GD_Maps` are `window`, the size of the filtering window (see Section 3.4.3), `nloop`, the number of times the image is filtered, `img_cluster` and `img_density`, the names of the output PNG images, and `bitmap`, a two-dimensional 400×400 array representing the point process in a format suitable for image processing.

Before describing `bitmap`, let's quickly summarize the observed data. It consists of a simulation of m superimposed stretched shifted Poisson-binomial point processes $P_1, \cdots, P_m$, as described in Exercise 18 and Sections 1.5.3, 1.5.4 and 3.4. An observed point $(x, y) = (X_{ih}, Y_{ik})$ in the state space is a point such that $(x, y) \in P_i$, and (h, k) is the index in the index space, with $h, k \in \{-n, \dots, n\}$. I used $n = 25$ and $m = 5$ in `PB_NN.py`, the parent Python script that calls `GD_Maps`. For the simulation, see source code `PB_NN.py` (Section 6.4, Part 2), or Formulas (8) and (9).

Now I can describe `bitmap`. Initially, `bitmap[pixelX][pixelY]=255`, unless there is a point of the process, say $(x, y) \in P_i$, such that `pixelX`=$\lfloor 200 \times (x \bmod \frac{2}{\lambda}) \rfloor$ and `pixelY`=$\lfloor 200 \times (y \bmod \frac{2}{\lambda}) \rfloor$. In that case, `bitmap[pixelX][pixelY]=processID`, where `processID` is the variable representing $i - 1$ in the source code. The brackets represent the floor function (also called integer function).

```
import math
from PIL import Image, ImageDraw # ImageDraw to draw rectangles etc.

def GD_Maps(method,bitmap,Nprocess,window,nloop,height,width,img_cluster,img_density):

  # PART 1: Allocate first image (clustering), including colors (palette)

  img1 = Image.new( mode = "RGBA", size = (width, height), color = (0, 0, 0) )
  pix1 = img1.load() # pix[x,y]=col[n] to modify the RGB color of a pixel
  draw1 = ImageDraw.Draw(img1,"RGBA")

  col1=[]
  col1.append((255,0,0,255))
  col1.append((0,0,255,255))
  col1.append((255,179,0,255))
  col1.append((0,0,0,255))
  col1.append((0,179,0,255))
  for i in range(Nprocess,256):
    col1.append((255,255,255,255))
  oldBitmap = [[255 for k in range(height)] for h in range(width)]
  densityMap= [[0.0 for k in range(height)] for h in range(width)]
  for pixelX in range(0,width):
    for pixelY in range(0,height):
      processID=bitmap[pixelX][pixelY]
      pix1[pixelX,pixelY]=col1[processID]
  draw1.rectangle((0,0,width-1,height-1), outline ="black",width=1)
  fname=img_cluster+'.png'
  img1.save(fname)
```

Part 2 performs supervised clustering in bitmap by filtering the entire bitmap `nloop` times. It also creates and filters densityMap, another bitmap with same dimensions, this time for unsupervised clustering, and using a slightly different filter. Both filters take place within the same loop. The contribution $g(u,v)$ of a point (u,v), in the small moving window (the local filter), is function of its distance to the center of the window: $g(u,v) = (1+u^2+v^2)^{-1/2}$. Also, for unsupervised clustering, successive applications of the filter to the entire image are increasingly dampened. The purpose is to get the algorithm to converge to a meaningful solution. For details, see Section 3.4. Finally, boundary effects are taken care of.

```
# PART 2: Filter bitmap and densityMap

for loop in range(nloop): #

  print("loop",loop,"out of",nloop)
  for pixelX in range(0,width):
    for pixelY in range(0,height):
      oldBitmap[pixelX][pixelY]=bitmap[pixelX][pixelY]

  for pixelX in range(0,width):
    for pixelY in range(0,height):
      count=[0] * Nprocess
      density=0
      maxcount=0
      topProcessID=255 # dominant processID near (pixelX, pixelY)
      for u in range(-window,window+1):
        for v in range(-window,window+1):
          x=pixelX+u
          y=pixelY+v
          if x<0:
            x+=width   # boundary effect correction
          if y<0:
            y+=height  # boundary effect correction
          if x>=width:
            x-=width   # boundary effect correction
          if y>=height:
            y-=height  # boundary effect correction
          if method == 0:
            dist2=1
          else:
            dist2=1/math.sqrt(1+u*u + v*v)
          processID=oldBitmap[x][y]
          if processID < 255:
            count[processID]=count[processID]+dist2
            if count[processID]>maxcount:
              maxcount=count[processID]
              topProcessID=processID
            density=density+dist2
      density=density/(10**loop) # 10 at power loop (dampening)
      densityMap[pixelX][pixelY]=densityMap[pixelX][pixelY]+density
      bitmap[pixelX][pixelY]=topProcessID
```

Part 3 assigns the right color to each pixel of the supervised clustering image and generates the associated PNG output file. It also generates a highly granular histogram of the density values observed in the unsupervised clustering image. The histogram, used in Part 4, is stored in the hash table densityCountHash.

```
  # PART 3: Some pre-processing; output cluster image

  densityCountHash={} # use to rebalance gray levels
  for pixelX in range(0,width):
    for pixelY in range(0,height):
      topProcessID=bitmap[pixelX][pixelY]
      density=densityMap[pixelX][pixelY]
      if density in densityCountHash:
        densityCountHash[density]=densityCountHash[density]+1
      else:
        densityCountHash[density]=1
      pix1[pixelX,pixelY]=col1[topProcessID]

  draw1.rectangle((0,0,width-1,height-1), outline ="black",width=1)
  fname=img_cluster+str(loop)+'.png'
  img1.save(fname)
```

About the cluster image (see right plot in Figure 19): each point in the state space (modulo $2/\lambda$) colored in red, must be assigned to the red cluster, or in other words, classified to red. The same applies to the other colors, and the assignment mechanism is extremely fast. Each color is attached to one of the individual processes of the model; each process (its points generated via simulation) can be seen as a particular cluster of a training set (see right plot in Figure 17). So the code performs a very fast supervised clustering of the entire state space; the clustering algorithm is represented by the cluster image itself. See Section 3.4 for details.

Part 4 equalizes the density levels in the unsupervised clustering image, then allocates the image, allocates the gray levels in the palette, and save the density image (corresponding to unsupervised clustering) as a PNG file. I manually selected the thresholds in the equalizer algorithm for best visual impact; this needs to be automated. The result of the equalizer is this: the darkest areas in the image (left plot, Figure 19) correspond to the highest concentration of points in the state space modulo $2/\lambda$. This is where the mass of each cluster is concentrated. The cluster centers (darkest in the image) are the estimators of the shift vectors used to build the superimposed point process (one shift vector per individual process). The unsupervised clustering is performed on the observed data shown in the right plot in Figure 17, assuming the colors are not known. Detailed explanations are in Section 3.4.

```
# PART 4: Equalize gray levels in the density image; output image as a PNG file
# Also try https://www.geeksforgeeks.org/python-pil-imageops-equalize-method/

densityColorHash={}
col2=[]
size=len(densityCountHash) # number of elements in hash
counter=0

for density in sorted(densityCountHash):
  counter=counter+1
  quant=counter/size # always between zero and one
  if quant < 0.08:
    densityColorHash[density]=0
  elif quant < 0.18:
    densityColorHash[density]=30
  elif quant < 0.28:
    densityColorHash[density]=55
  elif quant < 0.42:
    densityColorHash[density]=90
  elif quant < 0.62:
    densityColorHash[density]=120
  elif quant < 0.80:
    densityColorHash[density]=140
  elif quant < 0.95:
    densityColorHash[density]=170
  else:
    densityColorHash[density]=254

# allocate second image (density image)

img2 = Image.new( mode = "RGBA", size = (width, height), color = (0, 0, 0) )
pix2 = img2.load() # pix[x,y]=col[n] to modify the RGB color of a pixel
draw2 = ImageDraw.Draw(img2,"RGBA")

# allocate gray levels (palette)
for i in range(0,256):
    col2.append((255-i,255-i,255-i,255))

# create density image pixel by pixel
for pixelX in range(0,width):
      for pixelY in range(0,height):
        density=densityMap[pixelX][pixelY]
        color=densityColorHash[density]
        pix2[pixelX,pixelY]=col2[color]

    # output density image
    draw2.rectangle((0,0,width-1,height-1), outline ="black",width=1)
    fname=img_density+str(loop)+'.png'
    img2.save(fname)

  return()
```

Conclusion We accomplished the whole purpose: estimating the unknown shift vectors (or cluster centers) associated to the observations, and inventing a new, very fast clustering technique (supervised or unsupervised) that can be performed in GPU [Wiki]. See also [29] (available online, here) for a similar use of GPU in the context of nearest neighbor clustering.

6.7 Source Code: Production of the Videos

The purpose here is to show you how to build high quality data videos, also called data animations. Emphasis is on automation and ease of implementation. The material, besides its educational value, also illustrates interesting aspects of the theory presented in this textbook. In particular, the video featuring fractional supervised clustering is an example of a neural network in action: each frame represents an hidden layer. It also illustrates machine learning performed in GPU (graphics processing unit) using image filtering techniques.

6.7.1 Dirichlet Eta Function

On GitHub: `av_demo.r`. R code using the AV library [Wiki] to produce the video related to the Dirichlet eta function $\eta(z)$, with $z = \sigma + it \in \mathbb{C}$, based on Formula (20) for the X coordinate (the real part), and Formula (21) for the Y coordinate (the imaginary part). The input file `av_demo_vg2cb.txt`, with 20,000 rows, is generated in the `PB_inference.xlsx` spreadsheet, in columns `T` to `Y` in the `Video_Riemann` tab. The R code generates 1000 PNG images `av_demo001.png`, `av_dem002.png` and so on (the frames of the video) in the directory `c:/Users/vince/tex/`. Each frame is based on 20 consecutive rows from the input file. Each row of the input file has 6 fields:

- `k`: the index (the row number),
- `x`, `y`: the sum (a 2D vector) of the first k terms of the series defining $\eta(z)$,
- `x2`, `y2`: the sum (a 2D vector) of the first $k+1$ terms of the series defining $\eta(z)$,
- `col`: specifies the color used to draw the arrow between (x, y) and (x_2, y_2).

The output video is `av_demo_vg2cb.mp4`, with a 1200×800 resolution and 12 frames per second. The initial conditions, that is the parameters σ and t, are specified in the spreadsheet. The above description corresponds to the basic version. The new version allows you to draw two orbits at the same time, that is, to work with two sets of values for σ, t. This allows you to simultaneously compare the convergence for two different sets of initial conditions. In that case, the first 10,000 rows correspond to the first set, and the other 10,000 to the second one. For more information, see Sections 2.3.2 and 2.4.1.

The RGB (red/green/blue) color system attached to the `arrows` R function relies on sinusoids: $0.9 \times |\sin(0.00100 \times \text{col})|$ for red, $0.6 \times |\sin(0.00075 \times \text{col})|$ for green, and $|\sin(0.00150 \times \text{col})|$ for blue. Note that in this data set, `col` and `k` are identical. The parameters in the sine functions have simple ratios, creating periodicity and harmonic waves [Wiki]. They are responsible for the harmonious display of colors.

```
# install.packages('Cairo')
library('Cairo');

CairoPNG(filename = "c:/Users/vince/tex/av_demo%03d.png", width = 1200, height = 800);
# https://www.rdocumentation.org/packages/Cairo/versions/1.5-14/topics/Cairo

data<-read.table("c:/Users/vince/tex/av_demo_vg2cb.txt",header=TRUE);

k<-data$k;
x<-data$x;
y<-data$y;
```

```
x2<-data$x2;
y2<-data$y2;
col<-data$col;

for (n in 1:1000) {

   plot(x,y,pch=20,cex=0,col=rgb(0,0,0),xlab="",ylab="",axes=FALSE );
   rect(-60, -60, 90, 30, density = NULL, angle = 45,
     col = rgb(0,0,0), border = NULL);
   # You need to adjust the size of the rectangle to your data

   a<-x[k <= n*20];
   b<-y[k <= n*20];
   a2<-x2[k <= n*20];
   b2<-y2[k <= n*20];
   c<-col[k <= n*20];
   arrows(a, b, a2, b2, length = 0, angle = 10, code = 2,
     col=rgb( 0.9*abs(sin(0.00100*col)),0.6*abs(sin(0.00075*col)),
     abs(sin(0.00150*col)) ));
}
dev.off();

png_files <- sprintf("c:/Users/vince/tex/av_demo%03d.png", 1:1000)
av::av_encode_video(png_files, 'c:/Users/vince/tex/av_demo_vg2cb.mp4', framerate = 12)
```

6.7.2 Fractal Supervised Clustering

On GitHub: PB_clustering_video.py. This code is a minimalist version of the PB_NN.py / GD_util.py combination, with both blended together. It does not compute nearest neighbor distances, does not perform unsupervised clustering, and does not use the equalizer. Rather it focuses on supervised clustering only, using a very small (atomic) filter window and a large number of passes (the variable nloop, set to 250). This allows you to create a large number of frames for the video. In a nutshell, the code generates 251 PNG images named img_0.png (the initial image corresponding to the training set), img_1.png, img_2.png and so: these are the frames of the video. Once created, they are combined into an mp4 video (in this case imgPB.mp4), for instance using the last two lines of the R code in Section 6.7.1. The methodology for the clustering algorithm is discussed in Sections 2.4.2 and 3.4.3.

Part 1 creates the training set consisting of 4 groups, via simulation. The variable ProcessID represents the group label. It is transformed into an image and stored in memory as bitmap (a 2D array), for easy image processing.

```
# PB_clustering_video.py

import math
import random
from PIL import Image, ImageDraw # ImageDraw to draw rectangles etc.
import moviepy.video.io.ImageSequenceClip # to produce mp4 video

Nprocess=4   # number of processes in the process superimposition
seed=82431  # arbitrary number
random.seed(seed) # initialize random generator
s=0.15 # scaling factor
shiftX=[]
shiftY=[]

for i in range(Nprocess) :
  shiftX.append(random.random())
  shiftY.append(random.random())
processID=0
height,width = (600, 600)
bitmap = [[255 for k in range(height)] for h in range(width)]

for h in range(-25,26):
  for k in range(-25,26):
    for processID in range(Nprocess):
      ranx=random.random()
      rany=random.random()
      ranID=random.random()
      if ranID < 0.20:
        processID=0
      elif ranID < 0.60:
        processID=1
      elif ranID < 0.90:
        processID=2
      else:
        processID=3
      x=shiftX[processID]+h+s*math.log(ranx/(1-ranx))
      y=shiftY[processID]+k+s*math.log(rany/(1-rany))
      if x>-3 and x<3 and x>-3 and x<3:
        xmod=1+x-int(x) # x modulo 2/lambda
        ymod=1+y-int(y) # y modulo 2/lambda
        pixelX=int(width*xmod/2)
        pixelY=int(height*(2-ymod)/2) # pixel (0,0) at top left corner
        bitmap[pixelX][pixelY]=processID
```

Part 2 generates the first frame `img_0.png` corresponding to the training set stored in the the `bitmap` array.

```
img1 = Image.new( mode = "RGBA", size = (width, height), color = (255, 255, 255) )
pix1 = img1.load() # pix[x,y]=col[n] to modify the RGB color of a pixel
draw1 = ImageDraw.Draw(img1,"RGBA")

col1=[]
col1.append((255,0,0,255))
col1.append((0,0,255,255))
col1.append((255,179,0,255))
col1.append((0,179,0,255))
col1.append((0,0,0,255))
for i in range(Nprocess,256):
  col1.append((255,255,255,255))

for pixelX in range(0,width):
  for pixelY in range(0,height):
      topProcessID=bitmap[pixelX][pixelY]
      pix1[pixelX,pixelY]=col1[topProcessID]

draw1.rectangle((0,0,width-1,height-1), outline ="black",width=1)
fname="img_0.png"
img1.save(fname)
```

Part 3 filters the `bitmap` image `nloop` times, generating the output frames `img_1.png`, `img_2.png` and so on, up to `img_251.png`.

```
nloop=250   # number of times the image is filtered

oldBitmap = [[255 for k in range(height)] for h in range(width)]
flist=[]

for loop in range(1,nloop+1):
  print("loop",loop,"out of",nloop+1)
  for pixelX in range(0,width):
    for pixelY in range(0,height):
      oldBitmap[pixelX][pixelY]=bitmap[pixelX][pixelY]
  for pixelX in range(1,width-1):
    for pixelY in range(1,height-1):
      x=pixelX
      y=pixelY
      topProcessID=oldBitmap[x][y]
      if topProcessID==255 or loop>50:
        r=random.random()
        if r<0.25:
          x=x+1
          if x>width-2:
            x=x-(width-2)
        elif r<0.5:
          x=x-1
          if x<1:
            x=x+width-2
        elif r<0.75:
          y=y+1
          if y>height-2:
            y=y-(height-2)
        else:
          y=y-1
          if y<1:
            y=y+height-2
        if loop>=50 and oldBitmap[x][y]==255:
          x=pixelX
          y=pixelY
      topProcessID=oldBitmap[x][y]
      bitmap[pixelX][pixelY]=topProcessID
      pix1[pixelX,pixelY]=col1[topProcessID]
  draw1.rectangle((0,0,width-1,height-1), outline ="black",width=1)
  fname="img_"+str(loop+1)+'.png'
  flist.append(fname)
  img1.save(fname)

clip = moviepy.video.io.ImageSequenceClip.ImageSequenceClip(flist, fps=20)
clip.write_videofile('img.mp4')
```

Glossary

m-interlacing	Superimposition of m Poisson-binomial processes, each with its own (shifted, stretched) *lattice space*, and its own intensity and scaling factor. See pages 11, 24, 34, 35, 36, 37, 38, 61, 63, 69, 75, 76
Anisotropy	A property of a point process: the points are evenly scattered in all directions. The point distribution is stochastically invariant under rotations. See pages 10, 37
Attraction / Repulsion	The larger the scaling factor s, the less repulsion (also called *inhibition*) among the points of the process. Cluster processes discussed here exhibit both attraction (points tend to cluster together) and repulsion among cluster centers (due to the underlying lattice structure). See pages 7, 16, 37
Boundary Effect	Also called edge effect. Bias in estimated point counts or nearest neighbor distances, due to unobserved points located outside but close to the finite window of observations. Also the result of missing points in the window of observations, in simulated point processes especially when the scaling factor s is large. Special techniques are used to handle this problem. See pages 10, 11, 12, 24, 25, 27, 29, 30, 32, 37, 38, 44, 46, 52, 60, 61, 64, 67, 76
Confidence Region	A confidence region of level γ is a 2-D set of minimum area covering a proportion γ of the mass of a bivariate probability distribution. See pages 24, 27, 32, 66, 68
Connected Component	A set of vertices in a graph that are connected to each other by paths. See also nearest neighbor graph. See pages 12, 37, 38, 48, 61, 63, 65, 69, 77, 78
Empirical Distribution	Cumulative frequency histogram attached to a statistic (for instance, nearest neighbor distances), and based on observations. When the number of observations tends to infinity and the bin sizes tends to zero, this step function tends to the theoretical cumulative distribution function of the statistic in question. See pages 8, 17, 24, 31, 33, 38, 46, 48, 50, 54, 64
Ergodicity	A statistic such as the interarrival times is ergodic if it has the same asymptotic distribution, whether it is computed on many observations from a single realization of the process, or averaged across many realizations, each with few observations. See pages 9, 32, 33, 37, 48, 59
Homogeneity	A property of a point process, characterized by an homogeneous intensity function, that is, constant or independent of the location. See pages 9, 10, 11, 61
Identifiability	A models is identifiable if it is uniquely defined by its parameters. Then it is possible to estimate each parameter separately. A trivial example of non-identifiability is when we have two parameters, say α, β, but they only occur in a product $\alpha\beta$. In that case, if $\alpha\beta = 6$, it is impossible to tell whether $\alpha = 2, \beta = 3$ or $\alpha = 1, \beta = 6$. See pages 10, 15, 24, 33, 34, 61
Index Space	Consists of the indices $h, k \in \mathbb{Z}$, attached to the points X_k in one dimension, or (X_h, Y_k) in two dimensions. See pages 6, 10, 11, 31, 44, 46, 52, 76, 82
Intensity	Core parameter of the Poisson-binomial process. Denoted as λ. It represents the granularity of the underlying lattice, that is, the point density. In d dimensions, $\mathrm{E}[N(B)] = 1$ for any hypercube B of length $1/\lambda$. Here N is the point count. When λ is constant (not depending on the location), the process is homogeneous. See pages 6, 15, 23, 25, 32, 37, 38, 40, 41, 61, 69
Interarrival Time	In one dimension, random variable measuring the distance between a point of the process and its closest neighbor to the right, on the real axis. Interarrival times are also called *increments*. See pages 7, 9, 25, 32, 33, 37, 41, 49, 59, 60, 64, 69, 71
Lattice Space	In two dimensions, it consists of the locations $(h/\lambda, k/\lambda)$ with $h, k \in \mathbb{Z}$. The distribution of a point (X_h, Y_k) is centered at $(h/\lambda, k/\lambda)$. The concept can be extended to any dimension. See pages 6, 10, 11, 12, 16, 35, 36, 46, 47, 61, 62, 63, 78
Location-scale	A random variable X has a location-scale distribution with two parameters, the scale s and location μ, if any linear transformation $a + bX$ has a distribution of the same family, with parameters respectively $b^2 s$ and $\mu + a$. Here μ is the expectation and s is proportional to the variance of the distribution. See pages 6, 10

Modulo Operator	Sometimes, it is useful to work with point “residues” modulo $\frac{1}{\lambda}$, instead of the original points, due to the nature of the underlying lattice. It magnifies the patterns of the point process. By definition, $X_k \bmod \frac{1}{\lambda} = X_k - \frac{1}{\lambda}\lfloor \lambda X_k \rfloor$ where the brackets represent the integer part function. See pages 36, 38, 40, 63, 76, 78, 84
NN Graph	Nearest neighbor graph. The vertices are the points of the process. Two vertices (the points they represent) are connected if at least one of the two points is nearest neighbor to the other one. This graph is undirected. See pages 22, 37, 65, 69, 78, 89
Point Count	Random variable, denoted as $N(B)$, counting the number of points of the process in a particular set B, typically an interval $[a, b]$ in one dimension, and a square or circle in two dimensions. See pages 5, 7, 24, 30, 32, 33, 37, 38, 44, 49, 59, 60, 62, 64, 67, 69, 71
Point Distribution	Random variable representing how a point of the process is distributed in a domain B; for instance, for a stationary Poisson process, points are uniformly distributed on any compact domain B (say, an interval in one dimension, or a square in two dimensions). See pages 7, 25, 37, 76
Quantile function	Inverse of the cumulative distribution function (CDF) F, denoted as Q. Thus if $P(X < x) = F(x)$, then $P(X < Q(x)) = x$. See pages 6, 13, 14, 38, 54, 57, 65
Scaling Factor	Core parameter of the Poisson-binomial process. Denoted as s, proportional to the variance of the distribution F attached to the points of the process. It measures the level of repulsion among the points (maximum if $s = 0$, minimum if $s = \infty$). In d dimensions, the process is stationary Poisson of intensity λ^d if $s = \infty$, and coincides with the fixed *lattice space* if $s = 0$. See pages 5, 6, 10, 11, 12, 15, 23, 25, 32, 37, 48, 59, 61, 62, 63, 64, 69, 73, 75
Shift vector	The lattice attached to a 2-D Poisson-binomial process consists of the vertices $(\frac{h}{\lambda}, \frac{k}{\lambda})$ with $h, k \in \mathbb{Z}$. A shifted process has its lattice translated by a shift vector (u, v). The new vertices are $(u + \frac{h}{\lambda}, v + \frac{k}{\lambda})$. See page 11, 36, 38, 40, 41, 63, 75, 84
Standardized Process	Poisson-binomial process with intensity $\lambda = 1$, scaling factor $s = 1$, and shifted (if necessary) so that the lattice space coincides with $\mathbb{Z}$ or $\mathbb{Z}^2$. See page 10
State Space	Space where the points of the process are located. Here, $\mathbb{R}$ or $\mathbb{R}^2$. See also *index space* and *lattice space*. See pages 6, 16, 23, 32, 36, 37, 38, 40, 44, 46, 51, 63, 82, 84
Stationarity	Property of a point process: the point distributions in two sets of same shape and area, are identical. The process is stochastically invariant under translations. See pages 6, 8, 11, 24, 29, 63

List of Figures

1 Convergence to stationary Poisson point process of intensity λ 8
2 Four superimposed Poisson-binomial processes: $s = 0$ (left), $s = 5$ (right) 12
3 Radial cluster process $(s = 0.2, \lambda = 1)$ with centers in blue; zoom in on the left 15
4 Radial cluster process $(s = 2, \lambda = 1)$ with centers in blue; zoom in on the left 16
5 Manufactured marble lacking true lattice randomness (left) 16
6 Locally random permutation σ; $\tau(k)$ is the index of X_k's closest neighbor to the right 17
7 Chaotic function (bottom), and its transform (top) showing the global minimum 18
8 Orbit of η in the complex plane (left), perturbed by a Poisson-binomial process (right) 21
9 Data animations – click on a picture to start a video 22
10 Minimum contrast estimation for (λ, s) 25
11 Confidence region for (p, q) – Hotelling's quantile function on the right 27
12 Period and amplitude of $\phi_\tau(t)$; here $\tau = 1, \lambda = 1.4, s = 0.3$ 29
13 Bias reduction technique to minimize boundary effects 29
14 A new test of independence (R-squared version) 30
15 Radial cluster process $(s = 0.5, \lambda = 1)$ with centers in blue; zoom in on the left 35
16 Radial cluster process $(s = 1, \lambda = 1)$ with centers in blue; zoom in on the left 35
17 Realization of a 5-interlacing with $s = 0.15$ and $\lambda = 1$: original (left), modulo $2/\lambda$ (right) 36
18 Rayleigh test to assess if a point distribution matches that of a Poisson process 39
19 Unsupervised (left) versus supervised clustering (right) of Figure 17 39

20 Elbow rule (right) finds $m = 3$ clusters in Brownian motion (left) 43
21 Elbow rule (right) finds $m = 8$ or $m = 11$ “jumps” in left plot 43
22 Each arrow links a point (blue) to its lattice index (red): $s = 0.2$ (left), $s = 1$ (right) 46
23 Distance between a point and its lattice location ($s = 1$) 47
24 Chaotic convergence of partial sums in Formula (19) 66

References

[1] Noga Alon and Joel H. Spencer. *The Probabilistic Method.* Wiley, fourth edition, 2016. 64

[2] José M. Amigó, Roberto Dale, and Piergiulio Tempesta. A generalized permutation entropy for random processes. *Preprint*, pages 1–9, 2012. arXiv:2003.13728. 17

[3] Luc Anselin. *Point Pattern Analysis: Nearest Neighbor Statistics.* The Center for Spatial Data Science, University of Chicago, 2016. Slide presentation. 13

[4] Adrian Baddeley. Spatial point processes and their applications. In Weil W., editor, *Stochastic Geometry. Lecture Notes in Mathematics*, pages 1–75. Springer, Berlin, 2007. 13

[5] Adrian Baddeley and Richard D. Gill. Kaplan-meier estimators of distance distributions for spatial point processes. *Annals of Statististics*, 25(1):263–292, 1997. 44

[6] David Bailey, Jonathan Borwein, and Neil Calkin. *Experimental Mathematics in Action.* A K Peters, 2007. 17

[7] N. Balakrishnan and C.R. Rao (Editors). *Order Statistics: Theory and Methods.* North-Holland, 1998. 47, 53, 64

[8] B. Bollobas and P. Erdös. Cliques in random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*, 80(3):419–427, 1976. 64

[9] Miklos Bona. *Combinatorics of Permutations.* Routledge, second edition, 2012. 17

[10] Jonathan Borwein and David Bailey. *Mathematics by Experiment.* A K Peters, 2008. 17

[11] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. Clustering and percolation of point processes. *Preprint*, pages 1–20, 2013. Project Euclid. 13

[12] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. On comparison of clustering properties of point processes. *Preprint*, pages 1–26, 2013. arXiv:1111.6017. 13

[13] Bartłomiej Błaszczyszyn and Dhandapani Yogeshwaran. Clustering comparison of point processes with applications to random geometric models. *Preprint*, pages 1–44, 2014. arXiv:1212.5285. 13

[14] Oliver Chikumbo and Vincent Granville. Optimal clustering and cluster identity in understanding high-dimensional data spaces with tightly distributed points. *Machine Learning and Knowledge Extraction*, 1(2):715–744, 2019. 43

[15] Yves Coudène. *Ergodic Theory and Dynamical Systems.* Springer, 2016. 9

[16] Noel Cressie. *Statistic for Spatial Data.* Wiley, revised edition, 2015. 13

[17] H.A. David and H.N. Nagaraja. *Order Statistics.* Wiley, third edition, 2003. 53

[18] Tilman M. Davies and Martin L. Hazelton. Assessing minimum contrast parameter estimation for spatial and spatiotemporal log-Gaussian Cox processes. *Statistica Neerlandica*, 67(4):355–389, 2013. 25

[19] Robert Devaney. *An Introduction to Chaotic Dynamical Systems.* Chapman and Hall/CRC, third edition, 2021. 9

[20] D.J.Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes – Volume I: Elementary Theory and Methods.* Springer, second edition, 2013. 13

[21] D.J.Daley and D. Vere-Jones. *An Introduction to the Theory of Point Processes – Volume II: General Theory and Structure.* Springer, second edition, 2014. 13

[22] David Coupier (Editor). *Stochastic Geometry: Modern Research Frontiers.* Wiley, 2019. 62

[23] Ding-Geng Chen (Editor), Jianguo Sun (Editor), and Karl E. Peace (Editor). *Interval-Censored Time-to-Event Data: Methods and Applications.* Chapman and Hall/CRC, 2012. 11

[24] Bradley Efron. Bootstrap methods: Another look at the jackknife. *Annals of Statistics*, 7(1):1–26, 1979. 24

[25] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, volume 5, pages 17–61, 1960. 64

[26] W. Feller. On the Kolmogorov-Smirnov limit theorems for empirical distributions. *Annals of Mathematical Statistics*, 19(2):177–189, 1948. 39, 64

[27] Peter J. Forrester and Anthony Mays. Finite size corrections in random matrix theory and Odlyzko's data set for the Riemann zeros. *Proceedings of the Royal Society A*, 471:1–21, 2015. arXiv:1506.06531. 22

[28] Guilherme França and André LeClair. Statistical and other properties of Riemann zeros based on an explicit equation for the n-th zero on the critical line. *Preprint*, pages 1–26, 2014. arXiv:1307.8395. 22

[29] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using GPU. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK, 2008. 40, 85

[30] Minas Gjoka, Emily Smith, and Carter Butts. Estimating clique composition and size distributions from sampled network data. *Preprint*, pages 1–9, 2013. arXiv:1308.3297. 64

[31] B.V. Gnedenko and A. N. Kolmogorov. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley, 1954. 42

[32] Michel Goemans and Jan Vondrák. Stochastic covering and adaptivity. In *Proceedings of the 7th Latin American Theoretical Informatics Symposium*, pages 532–543, Valdivia, Chile, 2006. 62

[33] M. Golzy, M. Markatou, and Arti Shivram. Algorithms for clustering on the sphere: Advances & applications. In *Proceedings of the World Congress on Engineering and Computer Science*, volume 1, pages 1–6, San Francisco, USA, 2016. 61

[34] R. Goodman. *Introduction to Stochastic Models*. Dover, second edition, 2006. 8

[35] Vincent Granville. Estimation of the intensity of a Poisson point process by means of nearest neighbor distances. *Statistica Neerlandica*, 52(2):112–124, 1998. 14

[36] Vincent Granville. *Applied Stochastic Processes, Chaos Modeling, and Probabilistic Properties of Numeration Systems*. Data Science Central, 2018. 9, 17, 42

[37] Vincent Granville. *Statistics: New Foundations, Toolbox, and Machine Learning Recipes*. Data Science Central, 2019. 25, 28, 39

[38] Vincent Granville, Mirko Krivanek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, 1996. 40

[39] Peter Hall. *Introduction to the theory of coverage processes*. Wiley, 1988. 62

[40] K. Hartmann, J. Krois, and B. Waske. *Statistics and Geospatial Data Analysis*. Freie Universität Berlin, 2018. E-Learning Project SOGA. 31

[41] Jane Hawkins. *Ergodic Dynamics: From Basic Theory to Applications*. Springer, 2021. 9

[42] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, 2002. 57

[43] Zhiqiu Hu and Rong-Cai Yang. A new distribution-free approach to constructing the confidence region for multiple parameters. *PLOS One*, 8(12), 2013. 28

[44] Aleksandar Ivić. *The Riemann's Zeta Function: Theory and Applications*. Dover, reprint edition, 2003. 22

[45] Timothy D. Johnson. Introduction to spatial point processes. *Preprint*, page 2008. NeuroImaging Statistics Oxford (NISOx) group. 13

[46] Richard Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(2):665–671, 1939. 62

[47] Michael A. Klatt, Jaeuk Kim, and Salvatore Torquato. Cloaking the underlying long-range order of randomly perturbed lattices. *Physical Review Series E*, 101(3):1–10, 2020. 53

[48] Denis Kojevnikov, Vadim Marmer, and Kyungchul Song. Limit theorems for network dependent random variables. *Journal of Econometrics*, 222(2):419–427, 2021. 13

[49] Samuel Kotz, Tomasz Kozubowski, and Krzystof Podgorski. *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*. Springer, 2001. 58

[50] K. Krishnamoorthy. *Handbook of Statistical Distributions with Applications*. Routledge, second edition, 2015. 73

[51] Faraj Lagum. *Stochastic Geometry-Based Tools for Spatial Modeling and Planning of Future Cellular Networks*. PhD thesis, Carleton University, 2018. 13

[52] Günther Last and Mathew Penrose. *Lectures on the Poisson Process*. Cambridge University Press, 2017. 13

[53] André LeClair. Riemann hypothesis and random walks: The zeta case. *Symmetry*, 13:1–13, 2021. 22

[54] G. Last M.A. Klatt and D. Yogeshwaran. Hyperuniform and rigid stable matchings. *Random Structures and Algorithms*, 2:439–473, 2020. 13

[55] J. Mateu, C. Comas, and M.A. Calduch. Testing for spatial stationarity in point patterns. In *International Workshop on Spatio-Temporal Modeling*, 2010. 9

[56] Jorge Mateu, Frederic P Schoenberg, and David M Diez. On distances between point patterns and their applications. *Preprint*, pages 1–29, 2010. 13

[57] Natarajan Meghanathan. Distribution of maximal clique size of the vertices for theoretical small-world networks and real-world networks. *Preprint*, pages 1–20, 2015. arXiv:1508.01668. 64

[58] Jesper Møller. Introduction to spatial point processes and simulation-based inference. In *International Center for Pure and Applied Mathematics (Lecture Notes)*, Lomé, Togo, 2018. 13, 25, 33

[59] Jesper Møller and Frederic Paik Schoenberg. Thinning spatial point processes into Poisson processes. *Random Structures and Algorithms*, 42:347–358, 2010. 10

[60] Jesper Møller and Rasmus P. Waagepetersen. *An Introduction to Simulation-Based Inference for Spatial Point Processes*. Springer, 2003. 13

[61] Jesper Møller and Rasmus P. Waagepetersen. *Statistical Inference and Simulation for Spatial Point Processes*. CRC Press, 2007. 13

[62] S. Ghosh N., Miyoshi, and T. Shirai. Disordered complex networks: energy optimal lattices and persistent homology. *Preprint*, pages 1–44, 2020. arXiv:2009.08811. 5

[63] Saralees Nadarajah. A modified Bessel distribution of the second kind. *Statistica*, 67(4):405–413, 2007. 58

[64] Melvyn B. Nathanson. *Additive Number Theory: The Classical Bases*. Springer, reprint edition, 2010. 63

[65] D Noviyanti and H P Lestari. The study of circumsphere and insphere of a regular polyhedron. *Journal of Physics: Conference Series*, 1581:1–10, 2020. 61

[66] Yosihiko Ogata. Cluster analysis of spatial point patterns: posterior distribution of parents inferred from offspring. *Japanese Journal of Statistics and Data Science*, 3:367–390, 2020. 13

[67] Vamsi Paruchuri, Arjan Durresi, and Raj Jain. Optimized flooding protocol for ad hoc networks. *Preprint*, pages 1–10, 2003. arXiv:cs/0311013v1. 62

[68] Yuval Peres and Allan Sly. Rigidity and tolerance for perturbed lattices. *Preprint*, pages 1–20, 2020. arXiv:1409.4490. 5, 13

[69] Brian Ripley. *Stochastic Simulation*. Wiley, 1987. 73

[70] Peter Shirley and Chris Wyman. Generating stratified random lines in a square. *Journal of Computer Graphics Techniques*, 6(2):48–54, 2017. 61

[71] Karl Sigman. Notes on the Poisson process. New York NY, 2009. IEOR 6711: Columbia University course. 9, 13

[72] Luuk Spreeuwers. *Image Filtering with Neural Networks: Applications and Performance Evaluation*. PhD thesis, University of Twente, 1992. 40

[73] J. Michael Steele. Le Cam's inequality and Poisson approximations. *The American Mathematical Monthly*, 101(1):48–54, 1994. 19, 52

[74] Dietrich Stoyan, Wilfrid S. Kendall, Sung Nok Chiu, and Joseph Mecke. *Stochastic Geometry and Its Applications*. Wiley, 2013. 62

[75] Anna Talgat, Mustafa A. Kishk, and Mohamed-Slim Alouini. Nearest neighbor and contact distance distribution for binomial point process on spherical surfaces. *IEEE Communications Letters*, 24(12):2659–2663, 2020. 61

[76] Gerald Tenenbaum. *Introduction to Analytic and Probabilistic Number Theory*. American Mathematical Society, third edition, 2015. 17

[77] Remco van der Hofstad. *Random Graphs and Complex Networks*. Cambridge University Press, 2016. 64

[78] Robert Williams. *The Geometrical Foundation of Natural Structure: A Source Book of Design*. Dover, 1979. 62

[79] Oren Yakir. Recovering the lattice from its random perturbations. *Preprint*, pages 1–18, 2020. arXiv:2002.01508. 13, 53

[80] Ruqiang Yan, Yongbin Liub, and Robert Gao. Permutation entropy: A nonlinear statistical measure for status characterization of rotary machines. *Mechanical Systems and Signal Processing*, 29:474–484, 2012. 17

[81] D. Yogeshwaran. Geometry and topology of the boolean model on a stationary point processes : A brief survey. *Preprint*, pages 1–13, 2018. Researchgate. 13

[82] Tonglin Zhang. A Kolmogorov-Smirnov type test for independence between marks and points of marked point processes. *Electronic Journal of Statistics*, 8(2):2557–2584, 2014. 30

Index

m-interlacing, 11, 24, 34–38, 40, 61, 63, 69, 75, 76
m-mixture, 35–38, 60, 61, 63

anisotropy, 10, 17, 37
attraction (point process), 7, 16
attractor (distribution), 38, 42, 47, 64

Berry-Esseen theorem, 67
Bessel function, 58
Beta function, 15
bias, 44
binomial distribution, 7, 38, 60
boundary effect, 10–12, 17, 24, 25, 27, 29, 30, 32, 34, 37, 38, 44, 46, 52, 60, 61, 64, 67, 76, 83
Brownian motion, 23, 41

Cauchy distribution, 42, 51
censored data, 11, 44, 60
central limit theorem, 26, 42, 52, 56
    multivariate, 67
chaotic convergence, 21, 65
characteristic function, 58, 67
chi-squared distribution, 67
child process, 13, 74
clique (graph theory), 64
cluster process, 11, 13, 36, 37
    on the sphere, 61
clustering, 40
    fractal clustering, 24, 70
    fuzzy, 24
    GPU-based, 24, 36
    supervised, 23, 36
    unsupervised, 23, 36
Cochran's theorem, 67
confidence band, 39
confidence interval, 32, 37
confidence level, 27
confidence region, 27, 32, 66, 68
    dual region, 24, 27, 68
connected components, 12, 37, 38, 48, 61, 63, 65, 69, 77, 78
contour line, 26
convergence acceleration, 65
convolution of distributions, 51, 57, 58
counting measure, 7
covariance matrix, 67
covering (stochastic), 62
cross-validation, 28

data animation, 24
degrees of freedom, 67
density estimation, 14
deviate, 73
Dirichlet eta function, 18, 20, 22, 43, 66, 69
distribution
    binomial, 7, 38, 60
    Cauchy, 42, 51, 52, 73
    chi-squared, 67
    empirical, 54
    exponential-binomial, 5, 49
    Fréchet, 23, 42
    Gaussian, 67
    generalized logistic, 5, 13–15, 33, 56, 65
    half-logistic, 15
    Hotelling, 26
    Laplace, 54, 55, 58, 65
    location-scale, 6, 10
    logistic, 11, 14, 73
    Lévy, 42
    metalog, 15
    modified Bessel, 58
    Poisson, 18
    Poisson-binomial, 5, 7, 13, 18, 47
    Rayleigh, 38, 47, 61
    stable distribution, 58
    triangular, 57
    truncated, 51, 57
    uniform, 53, 73
    Weibull, 23, 38, 42, 47
domain of attraction, 64
dual confidence region, 24, 27, 68
dynamical systems, 9, 23, 43, 48, 64

edge (graph theory), 63
edge effect (statistics), 11, 44
elbow rule, 11, 30, 36, 38, 41
empirical distribution, 8, 17, 24, 31, 33, 38, 46, 48, 50, 54, 64
entropy, 17
ergodicity, 9, 32, 33, 37, 48, 59
extreme values, 42, 46

filtering (image processing), 22–24, 40
fixed point algorithm, 48
Fourier transform, 58
fractal clustering, 22, 24, 70
fractal dimension, 43
Fréchet distribution, 23, 42

Gamma function, 42
Gaussian distribution, 67
    multivariate, 67
GPU-based clustering, 23, 24, 36, 40
graph, 12, 64
    connected components, 12, 37, 63, 69, 77, 78
    edge, 12
    nearest neighbor graph, 22, 37, 65, 69, 78
    node, 12, 64
    path, 12
    random graph, 64
        random nearest neighbor graph, 64
    undirected, 12, 37, 38, 63–65, 69, 78
    vertex, 12
graph theory, 12, 63
grid, 5, 6

hash table, 16, 69, 77
hexagonal lattice, 13
hidden model, 13, 16, 33, 46, 52
high precision computing, 28

histogram equalization, 40
homogeneity, 9, 11, 14, 38, 61
Hotelling distribution, 26, 67

identifiability, 10, 15, 24, 33, 34, 61
independent increments, 9, 37
index, 16, 28, 31, 52
  index discrepancy, 17
  index process, 16
  index space, 6, 11, 31, 44, 46, 52, 76, 82
inhibition (point process), 5
intensity function, 6, 14, 15, 23, 25, 32, 37, 38, 40, 41, 61, 69
interarrival times, 7, 9, 25, 32, 33, 37, 41, 49, 57, 59, 64, 69, 71
  standardized, 58
interlaced processes, 11, 36
inverse model, 13
inverse transform sampling, 14, 54, 65, 73

Kolmogorov-Smirnov test, 30, 64

Laplace distribution, 58
lattice, 5, 6, 10, 13
  Bravais lattice, 62
  congruent lattices, 63
  hexagonal, 13, 62
  lattice group (group theory), 63
  lattice index, 76
  lattice space, 10–12, 16, 35, 36, 46, 47, 61, 63, 78
  perturbed lattice, 5
  semi-regular, 63
  shifted, 12, 13
  stretched, 12
  vertex, 13
law of large numbers, 26
Le Cam's theorem, 13, 19, 52, 58
location-scale distribution, 6, 10
logistic map, 48
Lévy distribution, 42
Lévy flight, 42

Mahalanobis transformation, 11
marked point process, 10
metalog distribution, 15
minimum contrast estimation, 24, 25, 33
mixture model, 11, 36
model-free inference, 28
modulo operator (point processes), 36, 38, 40, 63, 76, 78, 84
moment generating function, 8, 15

nearest neighbors, 5, 32, 37, 44, 48, 50, 61, 64, 77
  nearest neighbor distances, 24, 35, 37, 39, 44, 47, 61, 63, 69, 75, 77
  nearest neighbor graph, 22, 37, 65, 69, 78
    random nearest neighbor graph, 64
neural network, 22–24, 40
normal distribution, 67
  multivariate, 67
numerical stability, 28, 48, 53, 57, 66

order statistics, 46, 53
outliers, 46
overfitting, 33

palette optimization, 24
parent process, 13, 37, 74
partition, 60
permutation
  entropy, 17
  random permutation, 13, 17
perturbed lattice process, 5
point count distribution, 5, 7, 24, 30, 32, 33, 37, 38, 44, 49, 59, 60, 62, 64, 67, 69, 71
point distribution, 7, 25, 37, 76
point process, 6
  anisotropic, 10
  attractive, 7
  binomial, 5, 7
  cluster process, 13
    child process, 13
    Matérn, 13
    Neyman-Scott, 13
    parent process, 13
  ergodic, 9
  intensity, 69
  interarrival times, 69
  interlaced, 11
  marked process, 10
  mixture, 11, 36, 60, 63
  non-homogeneous, 14
  perturbed lattice process, 5, 12, 13, 16
  point count distribution, 5, 60
  Poisson, 5, 18, 46, 51
    non homogeneous, 9
  Poisson-binomial, 5, 6, 18
  radial, 13
  renewal process, 9, 13
  repulsive, 7, 15
  shifted, 10, 36
  stationary, 6, 8
  stretched, 10, 12
  superimposed, 11, 36, 63
  thinned, 10
point process operations, 10, 13
Poisson distribution, 18
Poisson point process, 18
Poisson-binomial distribution, 18, 47
Poisson-binomial point process, 18
  standardized, 10
proxy space, 26
pseudo-random number generator, 28, 48

quantile, 14
  fundamental theorem, 15, 54
  quantile function, 6, 13, 14, 38, 54, 57, 65

radial distribution, 13, 36
random function, 13, 18, 20, 23
random graph, 64
random lines, 61
random numbers, 48

random permutation, 17
random walk, 42
Rayleigh distribution, 38, 47, 61
Rayleigh test, 38
records, 46
renewal process, 9, 13
repulsion (point process), 5, 7, 15, 39
resampling, 28, 39
Riemann hypothesis, 21, 22
Riemann zeta function, 13, 18, 20, 43

sample size, 27
scaling factor, 5, 6, 11, 12, 15, 23, 25, 32, 37, 44, 46, 48, 59, 61–64, 69, 73, 75
shift vector, 11, 36, 38, 40, 41, 63, 75, 84
shifted process, 10, 40, 41, 63
simulation, 27, 48
spatial process, 36
spatial statistics, 13
stable distribution, 42, 52, 58
standardized arrival times, 58
standardized point process, 10, 38
state space, 6, 16, 23, 32, 36–38, 40, 44, 46, 51, 82, 84
stationarity, 6, 8, 11, 24, 29, 59, 63
stochastic convergence, 24
stochastic geometry, 61, 62
stochastic residues, 36
stretching (point process), 10, 12, 38, 75
superimposition (point processes), 11, 36
symbolic math, 48

tessellation, 62
thinning (point process), 10
tiling (spatial processes), 63
training set, 23
transcendental number, 48
truncated distribution, 51, 57

vertex (graph theory), 12, 61, 63, 64
visualization, 12
Voronoi tessellation, 62

Weibull distribution, 23, 38, 42, 47
Wiener process, 41